PLATAFORMA REMOTA DE GESTÃO DE UM PARQUE DE GATEBOXES

Vítor Manuel Ribeiro Aires

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
2011

Este relatório satisfaz, parcialmente, os requisitos que constam da Ficha de Disciplina de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Eletrotécnica e de Computadores – Ramo Telecomunicações

Candidato: Vítor Manuel Ribeiro Aires, Nº 1060881, 1060881@isep.ipp.pt
Orientação científica: Prof. Doutor Jorge Mamede, jbm@isep.ipp.pt

Empresa: NextToYou - Network Solutions, Lda.

Supervisão: Eng.º Vítor Brandão, vbrandao@nexttoyou.pt

Departamento de Engenharia Eletrotécnica

Instituto Superior de Engenharia do Porto

16 de Novembro de 2011

Agradecimentos

Em primeiro lugar, quero dirigir o meu agradecimento ao Prof. Doutor Jorge Mamede, orientador do projeto no Instituto Superior de Engenharia do Porto (ISEP), pela orientação, sugestões e disponibilidade ao longo do projeto.

Na empresa NextToYou agradeço em especial ao meu orientador Eng.º Vítor Brandão, pela disponibilidade em orientar e fornecer as informações necessárias para a concretização do projeto.

O meu maior agradecimento é, para minha família, pelo apoio que me deram durante esta etapa da minha vida.

Agraço ainda aos meus colegas e amigos que estiveram sempre presentes.

A todos, o meu OBRIGADO!

Resumo

A monitorização de redes é um aspeto de elevada importância, principalmente em redes de média ou grande dimensão. A necessidade de utilização de uma ferramenta para realização dessa gestão facilita o trabalho e proporciona de uma forma mais rápida e eficaz a identificação de problemas na rede e nos seus sistemas. Neste sentido, o presente trabalho tem como objetivo o desenvolvimento de uma solução para a monitorização de GateBoxes, um dos produtos desenvolvidos e comercializados pela empresa NextToYou.

A necessidade de monitorização das GateBoxes, por parte da NextToYou, é essencial para que possa detetar falhas no seu funcionamento ou realizar notificações aquando da deteção de problemas para uma rápida resolução. Neste contexto a empresa decidiu implementar uma ferramenta para a referida monitorização e propôs, no âmbito da tese, o desenvolvimento de uma aplicação que satisfizesse esses propósitos. Disponibilizou então, para o desenvolvimento uma plataforma, a WebForge, e definiu alguns requisitos funcionais dessa ferramenta, tais como, a monitorização remota de informação, gestão de alarmes, geração de avisos e notificações.

Para a elaboração deste trabalho foram realizados estudos teóricos sobre o tema da gestão e monitorização remotas, realizando-se posteriormente o desenvolvimento de uma aplicação para a monitorização de GateBoxes. Após a implementação efetuou-se a validação do trabalho realizado através da execução de testes e demonstrações, de forma a poder validar e verificar o desempenho do sistema.

Palavras-Chave

Monitorização, Rede, GateBox, NextToYou.

Abstract

Network monitoring is an aspect of great importance especially in networks of medium or large dimension. The need to use a tool for management procedures makes the job easier and provides a quicker and more effective way to identify problems in the network and its systems. In that way, the current work has as an objective the development of a solution for monitoring GateBoxes, which are one kind product developed and commercialized by the NextToYou company.

Monitoring GateBoxes is essential for NextToYou, so that failures can be detected, or that notifications can be produced when problems occur, for a quick resolution. In this context the company decided to build a tool for monitoring those units and so, it proposed, as part of the thesis, the development of an application meeting their needs. For that, NextToYou provided the WebForge development platform and defined some functional requirement for the tool, such as, the remote monitoring of information, alarm management, warning generation and notifications.

For the development of this work, theoretical studies about the remote management and monitoring subjects were carried out, which conduct to the development of an application for monitoring GateBoxes. After the development, the tool evaluation was carried out through tests and demonstrations, so that the system's performance could be validated and verified.

Keywords

Monitoring, Network, GateBox, NextToYou.

Índice

Índice de Figuras

Índice de Tabelas

Acrónimos

API – Application Programming Interface

ASCE – Association control Service Element

ASN.1 – Abstract Syntax Notation

CMIP – Common Management Information

CPU – Central Processing Unit

CSS – Cascade Style Sheet

DES – Data Encryption Standard

GUI – Graphical User Interface

GPL – General Public License

HTTP – Hypertext Transfer Protocol

ID – Identifier

IE – Internet Explorer

IETF – Internet Engineering Task Force

ITED – Infra-estruturas de Telecomunicações em Edifícios

ITUR – Infra-estruturas de Telecomunicações em Urbanizações

IP – Internet Protocol

MD5 – Message-Digest algorithm 5

MIB – Management Information Base

MVC – Model-View-Controller

NMA – Network Management Application

OID – Object Identifier

OSI – Open Systems Interconnection

PCL – PHP Extension Community Library

PEAR – PHP Extension and Application Repository

PERL – Practical extraction and Report Language

PHP – Hypertext Preprocessor

POP3 – Post Office Protocol

RAM – Random Access Memory

ROSE – Remote Operation Service Element

RRDTool – Round Robin Database Tool

SHA – Secure Hash Algorithm

SLA – Service Level Agreement

SMI – Structure of Management Information

SMS – Short Message Service

SMTP – Simple Mail Transfer Protocol

SNMP – Simple Network Management Protocol

TCP – Transmission Control Protocol

UDP – User Datagram Protocol

1. INTRODUÇÃO

Atualmente, a utilização de tecnologias de comunicação e informação, principalmente, em redes e em sistemas de processamento distribuídos são de uma importância vital e crescente em todos os tipos de empresas [1]. A evolução da dimensão, heterogeneidade e complexidade dos sistemas distribuídos, obriga a que a conceção, realização e operação de uma rede contemple um conjunto de mecanismos e ferramentas, para a monitorização e controlo dos recursos de comunicação de forma a garantir a qualidade dos serviços [2]. Nas redes, a organização e gestão não pode ser realizada apenas com o esforço humano, sendo necessária a utilização de ferramentas automatizadas de monitorização de rede. A capacidade de visualização de informação, gráficos e relatórios, além de notificações que alertam o utilizador de eventuais anomalias no sistema monitorizado, proporciona ao gestor da rede o acompanhamento da rede em tempo real [3].

1.1. CONTEXTUALIZAÇÃO

O presente trabalho surge no âmbito da necessidade da empresa NextToYou monitorizar um dos produtos desenvolvidos e comercializados, a GateBox. Para isso, é necessário desenvolver uma aplicação Web que permita a interação com GateBoxes já instaladas. Essa aplicação deverá proporcionar a monitorização de avisos e alarmes gerados pelas GateBoxes e incluir um conjunto de agentes que permitam a configuração remota de determinadas funcionalidades.

A NextToYou é uma empresa de Telecomunicações que desenvolve um conjunto de sistemas modulares que integram diferentes tipos de soluções: comunicações, multimédia, segurança e gestão de edifícios. A GateBox é um desses sistemas de integração de serviços e de gestão de comunicações em edifícios residenciais ou empresariais e proporciona as seguintes funcionalidades [4]:

- Sistema de videovigilância integrado;
- Sistema integrado de videoporteiro com gravação de mensagens;
- Central telefónica digital interna no edifício, ou equiparado;
- Serviços de rede comunitários (ex. página Web de condomínio, partilha de documentos, etc.);
- Acesso interno aos serviços em qualquer divisão da habitação;
- Acesso sem fios aos serviços nas áreas comuns (e.g., jardim, piscina, sala de reuniões, etc.);
- Acesso externo aos serviços através de Internet.

1.2. OBJETIVOS

O objetivo principal deste projeto é o estudo, a especificação e a implementação de um sistema que permita a interação com unidades GateBoxes. Essa aplicação deverá possibilitar a monitorização de informações das GateBoxes, e também de avisos e alarmes gerados por estas. Dada a complexidade inerente a este objetivo, sentiu-se a necessidade de o subdividir em múltiplas tarefas de realização mais simples, tais como:

- Investigação, estudo e análise comparativa de plataformas utilizadas na administração remota de redes;
- Identificação e caracterização dos requisitos funcionais da aplicação administrativa;
- Implementação e desenvolvimento de um protótipo da aplicação;
- Demonstração do funcionamento, teste e análise de resultados.

1.3. ORGANIZAÇÃO DO RELATÓRIO

Este documento encontra-se estruturado em 6 capítulos. O capítulo 1 introduz este trabalho. O capítulo 2 aborda a monitorização de sistemas, com descrição de alguns protocolos e ferramentas utilizadas na gestão remota de sistemas de rede. No capítulo 3 serão apresentadas as especificações a aplicação desenvolver e apresentada a plataforma de desenvolvimento utilizada. No capítulo 4 são descritos todos os elementos do sistema GateKeeper[1]. No capítulo 5 serão apresentados alguns testes e demonstrações efetuados ao sistema GateKeeper. Por fim, no capítulo 6 serão reunidas as principais conclusões e perspetivados futuros desenvolvimentos.

[1] Plataforma Web desenvolvida como solução para a gestão das GateBoxes.

2. MONITORIZAÇÃO DE SISTEMAS

A comunicação de dados através de redes de computadores e a possibilidade de transmissão de dados, originou a criação de uma estrutura que desenvolveu as atuais grandes interligações [5]. As redes de comunicação atuais são constituídas por uma grande variedade de dispositivos que devem comunicar e partilhar serviços. A eficiência dos serviços prestados por uma rede está normalmente associada ao bom desempenho dos sistemas que a constituem [6]. Assim, a utilização de um *software* capaz de monitorizar uma rede tem vindo a ser muito útil e indispensável para o gestor. A capacidade de através de gráficos e relatórios, se poder controlar uma rede, além de notificações que podem alertar o utilizador de eventuais anomalias no sistema gerido, proporciona um acompanhamento do estado do sistema em tempo real. A sua utilização permite detetar e isolar anomalias, ou até mesmo levar a que estas se evitem [3].

Ao longo deste capítulo, será abordado a importância da monitorização de sistemas, e descritos alguns protocolos e ferramentas utilizados na gestão de redes.

2.1. MONITORIZAÇÃO DE REDES

Inicialmente, as redes de computadores foram concebidas como meio para partilha de dispositivos periféricos (impressoras, *drivers* de alta velocidade, etc.), que existiam apenas em ambientes académicos, governamentais e em grandes empresas [6]. Entretanto, com a rápida evolução das tecnologias de redes, na década de 80, juntamente com a redução de custos dos diversos recursos houve uma expansão das redes de computadores por toda a sociedade [3]. Com isto, as redes passaram a fazer parte do quotidiano das pessoas como uma ferramenta, que oferece serviços e recursos, e que permitem uma maior interação entre os utilizadores, e consequente um aumento de produtividade.

A evolução observada resultou num aumento da complexidade das redes originando novos serviços como: correio eletrónico, transferência de arquivos, Internet, entre outros. O desenvolvimento de redes com a integração de serviços como voz, vídeo e de dados, introduziram a necessidade por parte do gestor da rede de controlar o desempenho desses recursos, com uma elevada eficácia e precisão, tornando-se assim importante a garantia de uma qualidade nos serviços prestados [6]. Para isso, surge a necessidade de monitorização, que tem como principais objetivos: o controlo dos dispositivos da rede, o aumento da disponibilidade da rede, a redução da complexidade da monitorização e a redução de custos de operação e manutenção [3].

2.1.1. O PROCESSO DA MONITORIZAÇÃO DE REDE

A monitorização de rede pode ser definida como a coordenação de recursos físicos (modems, routers, etc.) e lógicos (protocolos) distribuídos numa rede, assegurando fiabilidade, tempos de reposta aceitáveis e de segurança da informação. O modelo clássico de monitorização pode ser divido em três etapas [7]:

- Recolha de dados: um processo, normalmente automático, que consiste na obtenção de informações dos sistemas monitorizados;

- Diagnóstico: consiste no tratamento e análise dos dados recolhidos de uma série de procedimentos (por intermédio de um operador ou não) com o intuito de determinar a causa do problema no dispositivo em questão;

- Ação ou Controlo: consiste numa ação ou controlo, do dispositivo, após o diagnóstico do problema.

2.2. Sistema de monitorização

Um sistema de monitorização de redes pode ser definido como um conjunto de ferramentas integradas de gestão e controlo, que oferece uma interface e que obtém informações do estado dos dispositivos ou da rede, ou ambos. Para além de oferecer um conjunto de funcionalidades (comandos) que proporcionam a execução de atividades de monitorização no sistema.

A arquitetura geral dos sistemas de monitorização de redes é constituída por quatro componentes básicos: os elementos geridos, as estações de gestão, os protocolos e as informações de gestão. Nos elementos geridos/monitorizados deve existir um *software* agente que permite a monitorização e controlo do equipamento através da estação de gestão remota. Consoante a topologia da rede, esta pode ter uma estação de gestão (sistemas de monitorização centralizada) ou várias (sistemas de monitorização distribuída) para obtenção das informações dos dispositivos. Nestas estações encontra-se o *software* gestor, responsável pela comunicação com os agentes dos vários dispositivos. Para que possa ocorrer troca de informações é necessário um protocolo de gestão, que está encarregue das operações de monitorização [7].

A topologia utilizada era a de sistemas de monitorização centralizada, onde as funcionalidades de gestão se encontravam apenas numa estação de gestão. Contudo, com o crescimento das redes, tanto em tamanho, como em complexidade, os sistemas de monitorização baseados num único administrador tornaram-se inapropriados, devido ao grande volume de informação e à localização geográfica. Evidencia assim, a necessidade da distribuição da gestão por várias estações na rede [6].

2.2.1. Arquitetura do sistema de monitorização

Na Figura 1 pode-se visualizar uma representação básica da arquitetura de um sistema de monitorização de rede onde existe pelo menos uma estação de monitorização, que contém um conjunto de *software* denominado de *Network Management Application* (NMA), e que normalmente inclui uma interface para o utilizador autorizado gerir a rede. A NMA apresenta ao utilizador a informação do sistema quando efetuado o pedido (comandos). Essa comunicação é realizada através de um protocolo da camada de aplicação, específico para gestão de redes. Os outros componentes da rede, denominados agentes, respondem às solicitações da estação de gestão.

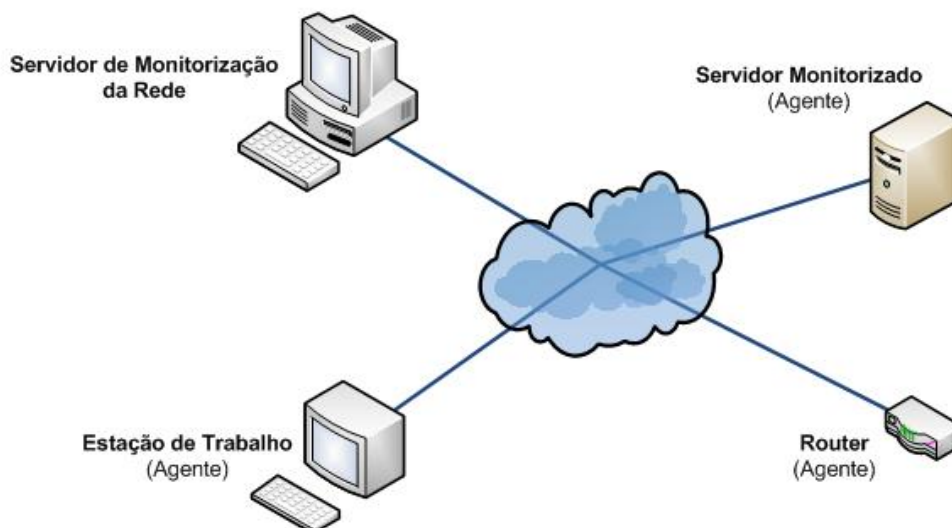



Figura 1 – Arquitetura de monitorização.

Para uma maior disponibilidade de gestão, normalmente são utilizados dois ou mais servidores de monitorização. Um deles é utilizado para o controlo do sistema, enquanto os outros ficam destinados a recolher estatísticas do sistema ou em estado de espera, em caso de problema no servidor que realize a gestão da rede [6].

2.3. PROTOCOLOS DE GESTÃO

Ao longo dos anos, vários investigadores tem trabalhado com o intuito de definir arquiteturas para a motorização de redes heterogéneas, ou seja, redes compostas por equipamentos de diferentes tecnologias. As dois principais protocolos de gestão de redes estão relacionadas com os módulos TCP/IP e OSI da ISO, denominadas respetivamente de *Simple Management Protocol* (SNMP) e *Common Management Information Protocol* (CMIP) [7].

O SNMP é o protocolo mais implementado atualmente pelas ferramentas de monitorização, que através de uma arquitetura gestor/agente proporciona uma troca de informações entre o administrador de uma rede e os dispositivos monitorizados. Essas informações podem ser relativas a problemas, erros, estado dos dispositivos, entre outras condições excecionais [7]. Da mesma forma que o SNMP, o CMIP é um protocolo de gestão com uma arquitetura gestor/agente. Este proporciona um conjunto de padrões de elevada complexidade para a realização da gestão de redes. Contudo, este protocolo não é muito utilizado devido à sua complexidade e lentidão na monitorização [40].

2.3.1. *SIMPLE NETWORK MANAGEMENT PROTOCOL* (SNMP)

O protocolo SNMP é atualmente, o protocolo de gestão mais utilizado em todo o mundo, tendo sido desenvolvido pelo *Internet Engineering Task Force* (IETF) [42], para monitorização de dispositivos numa rede IP [5]. Este protocolo permite que, através de simples operações, os seus utilizadores possam monitorizar dispositivos remotamente. Desde a sua criação tem vindo a sofrer evoluções que culminaram com as versões SNMPv2 e SNMPv3 [9].

O modelo de gestão da rede SNMP inclui os seguintes elementos-chave:

- Estação de gestão;
- Agente;
- Base de informação de gestão (*Management Information Base* (MIB));
- Protocolo de gestão de rede.

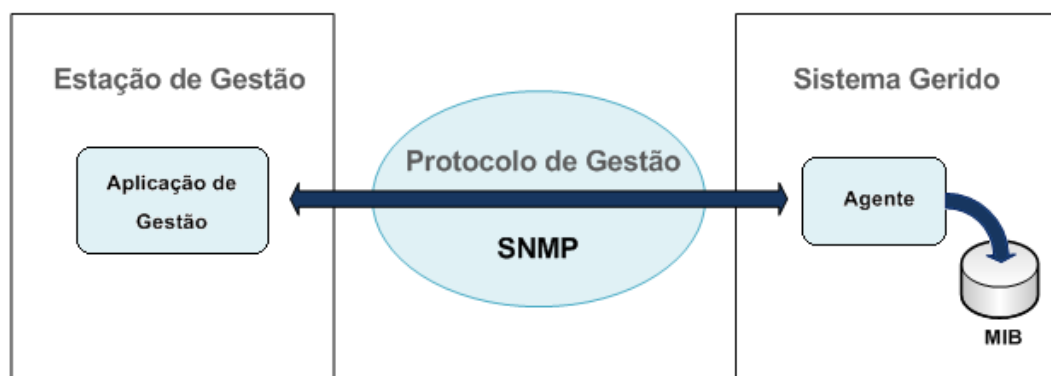



Figura 2 – Arquitetura do protocolo SNMP.

A estação de gestão e o agente, para poderem comunicar utilizam um protocolo de gestão. O SNMP é um protocolo que possibilita mecanismo simples de comunicação, que normalmente funciona sobre o protocolo *User datagram protocol* (UDP), podendo também funcionar sobre *Transmission Control Protocol* (TCP). A vantagem de ser utilizado sobre UDP é a de não receber a confirmação (*Acknowledge*) dos pedidos de informação efetuados, o que reduz a sobrecarga protocolar. Uma vez que o SNMP é um protocolo de pergunta e resposta, não é necessário o uso de mecanismos de deteção de erros, nem de testes à fiabilidade da comunicação (implementados pelo TCP) [10].

A estação de gestão é um dispositivo que realiza a interface entre o gestor de rede humano e o sistema de gestão da rede. Além disso, deve possuir no mínimo [11]:

- Um conjunto de aplicações de gestão para análise de dados, recuperação de falhas, entre outros;
- Na interface, o gestor de rede seja capaz de monitorizar e controlar todos os elementos da rede;
- Capacidade de traduzir as necessidades na monitorização e controlo real dos elementos remotos da rede, do gestor de rede;
- Uma base de dados de informação de monitorização de rede capaz de armazenar informações extraídas de todas as entidades geridas na rede.

O outro elemento ativo, e o mais importante no sistema de gestão de rede são os agentes. O agente responde a pedidos de informação requisitados pela estação de gestão, ou enviando para a estação de gestão informações importantes sobre o sistema sem que estas sejam solicitadas. O agente também define as políticas de acesso a gestores externos.

Para identificação dos recursos de um sistema, é representado para cada recurso um *Object Identifier* (OID). O OID apresenta-se basicamente como um bloco de informação de um recurso do sistema. Estes podem ser parâmetros de informação, estados do sistema, estatísticas de desempenho, entre outros. O conjunto de OIDs é fornecido e identificado numa estrutura denominada *Management Information Base* (MIB). Esta mantém-se atualizada com informações do sistema através do agente SNMP. A informação pode ser solicitada remotamente por uma estação de gestão ao agente, com a realização de pedidos (comandos). O agente consulta a MIB através do OIDs e envia a informação à estação de gestão que o solicitou [11].

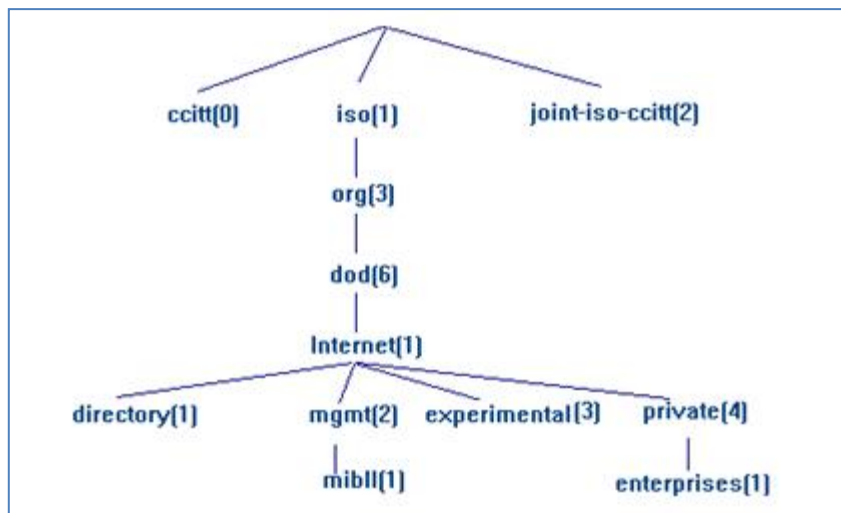



Figura 3 – Estrutura de uma MIB [12].

O SNMP tem como funcionalidades principais [11]:

- **Get**: Permite à estação de gestão obter um certo valor de um OID de um agente.
- **Set**: Permite à estação de gestão definir um certo valor de um OID de um agente.
- **Notify**: Permite ao agente notificar a estação de gestão de alguma ocorrência importante.

Os sistemas geridos devem implementar os protocolos SNMP, UDP e *Internet Protocol* (IP), bem como ter um agente de gestão que efetue a manutenção das informações da MIB do sistema e que gere respostas quando solicitadas pela estação de gestão.

2.3.1.1 SNMPv1

A primeira versão do protocolo SNMP, apenas suportava quatro operações: *GetRequest*, *SetRequest*, *GetNextRequest* e *Trap* [11].

A operação *GetRequest* permite que uma aplicação na estação de gestão efectue leituras dos valores dos objetos de MIB situada num agente. Cada *GetRequest* enviado através do gestor é correspondido pelo agente com um *GetResponse* com o mesmo identificador do pedido e com os valores correspondentes. Caso esta operação falhe, a mensagem de resposta enviará um código que identifica o tipo de erro ocorrido. O princípio de funcionamento da operação *GetNextRequest* é idêntico ao *GetRequest*, pois pode-se obter o valor do objeto seguinte na MIB, sendo este útil quando não se conhece a estrutura da MIB ou para ler valores de objetos sequencialmente.

A operação *SetRequest* (operação de escrita) permite que o gestor de rede coloque um certo valor num determinado objeto da MIB do agente. Após o sucesso ou insucesso dessa operação, o agente envia uma notificação para a estação de gestão, através de uma mensagem *GetResponse*. A ocorrência de falha nesta operação pode ser originada devido à inexistência do objeto na MIB, ao objeto ter apenas permissão de leitura ou ao valor pretendido não ser o correto para esse objeto.

Quando um agente necessitar de notificar a estação de gestão de um acontecimento relevante, ou configurado como tal pelo gestor, este pode-o fazer através da operação *Trap*. Esta notificação é assíncrona, não é solicitada pelo gestor, podendo ser enviada pelo agente em qualquer altura, no caso de ocorrer algum evento (erro, problema, entre outros) [14]. Na Figura 4 pode-se visualizar a sequência das operações acima referenciadas.

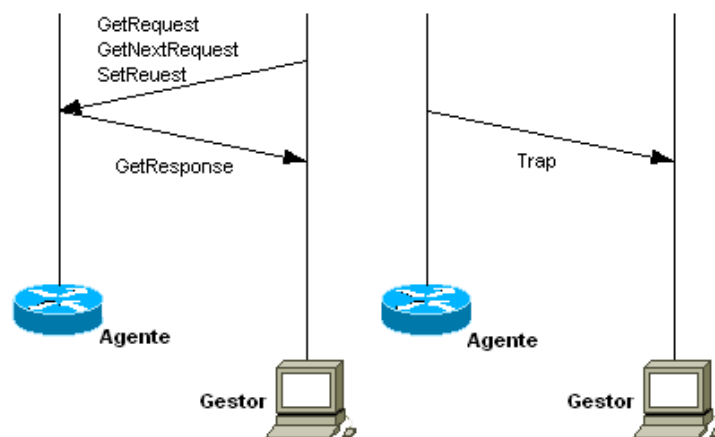



Figura 4 – Operações SNMP [13].

O SNMPv1, como primeira versão, teve falhas em termos de funcionamento em redes de grandes dimensões, onde existe um volume elevado de dados, com consequências de ineficiência e escalabilidade. Outra falha importante é em termos de segurança dado que esta era praticamente inexistente. Estas falhas foram tidas em atenção nas versões do protocolo desenvolvidas posteriormente [11].

2.3.1.2 SNMPv2

As limitações e problemas associados à primeira versão do protocolo SNMP levaram ao desenvolvimento de uma nova versão, o SNMPv2 [11]. Esta nova versão do protocolo contém melhorias na comunicação, na segurança, na organização da estrutura da informação de gestão (*Structure of management information* (SMI)), e permite interações de gestor para gestor. Implementa ainda duas novas operações: *GetBulkRequest* e *InformRequest*.

A operação GetBulkRequest permitiu corrigir umas das lacunas da primeira versão do protocolo, permitindo que através de uma única operação se possa ler uma tabela inteira de objetos de uma MIB. No caso de erro numa variável, ao efetuar uma leitura de mais do que uma variável, não é rejeitado todo o pedido, e são fornecidos os valores de todas as variáveis corretas [14].

Apesar de todos os melhoramentos na versão SNMPv2, houve uma lacuna em termos de segurança, e assim os mecanismos utilizados não se revelaram eficazes, sendo corrigidos numa versão subsequente [11].

2.3.1.3 SNMPv3

As deficiências de segurança nas versões anteriormente desenvolvidas foram tratadas pelo SNMPv3. Esta versão veio acrescentar melhores mecanismos de segurança, principalmente em termos de autenticação, privacidade e controlo de acesso [9]. Esta segurança procura evitar a alteração das mensagens enviadas, podendo ainda criar uma barreira a elementos estranhos à execução de operações de controlo, efetuadas pela operação *SetRequest*. Para isso, o SNMPv3 introduziu no protocolo mecanismos de encriptação, *Data Encryption Standard* (DES), e mecanismos de autenticação, *Message-Digest algorithm 5* (MD5) e *Secure Hash Algorithm* (SHA) [14].

2.3.2. *COMMON MANAGEMENT INFORMATION PROTOCOL* (CMIP)

O CMIP é um protocolo, definido segundo o padrão do modelo OSI, utilizado na monitorização de informações de redes [3]. Este protocolo atua no Nível de Aplicação do modelo OSI e é orientado à ligação utilizando para isso os serviços disponibilizados pelo

Association Control Service Element (ACSE[2]), *Remote Operation Service Element* (ROSE[3]) e pelo serviço de apresentação [40].

Da mesma forma que o SNMP, o CMIP define como se efetuam as trocas de informação entre a estação de gestão e os agentes dos dispositivos a monitorizar como, se ilustra na Figura 5. O agente é o responsável pela manutenção das informações da MIB, pela resposta a pedidos de informação requisitados pela estação de gestão e pelo envio de notificações. A estação de gestão é a responsável por monitorar remotamente os vários dispositivos da rede, os seus recursos e serviços, através operações efetuadas com o agente desses dispositivos.

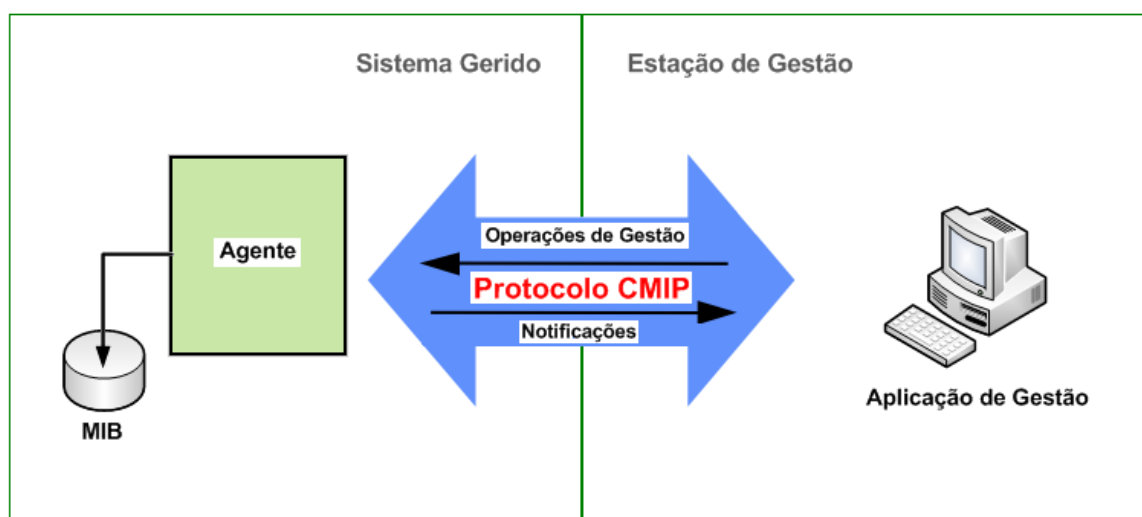



Figura 5 – Arquitetura do protocolo CMIP.

O tipo de informação a ser trocada entre as duas entidades (estação de gestão e agente) deve ter em conta o serviço *Common Mangement Information Service* (CMIS). O CMIS especifica um conjunto de serviços na monitorização, para realizarem operações de monitorização e de notificação [3].

O protocolo CMIP oferece as seguintes funcionalidades ao utilizador (gestor) [7]:

- **M-SET**: atribuição de valores a um objeto;
- **M-GET**: leitura de valores de objetos;

---

[2] Método da camada da aplicação do modelo OSI utilizado para estabelecer uma ligação entre duas aplicações. Este confirma as identidades e os contextos das aplicações, e ainda pode verificar a segurança através da autenticação [44].

[3] Método da camada de aplicação do modelo OSI que permite a realização de operações remotas entre aplicações [43]

- **M-ACTION**: execução de uma ação sobre um objeto;
- **M-CREATE**: criação de uma nova instância de uma classe de objetos;
- **M-DELETE**: eliminação de uma ou mais instâncias de objetos;
- **M-CANCEL-GET**: cancelamento de uma operação M-GET demorada;
- **M-EVENT-REPORT**: envio de notificações ao gestor da rede.

Como já foi referido anteriormente, o CMIS é uma norma que especifica conjunto de serviços de monitorização (aplicação de gestão e agente). Esses serviços dividem-se em três grupos [41]:

- **Serviços de associação**: são utilizados para estabelecer as associações necessárias para realização de ligações entre a estação de gestão e os dispositivos monitorizados. Sendo necessários para isso, os serviços oferecidos da aplicação ACSE.
- **Serviços de notificação**: são utilizados para o agente alertar a estação de gestão da ocorrência de um evento.
- **Serviços de operação**: são utilizados na estação de gestão (o gestor) para efetuar a recolha ou alteração de informação da MIB do agente.

Os serviços CMIS e o protocolo CMIP são utilizados para diferentes tarefas na implementação de sistemas de monitorização de uma rede baseada no modelo de comunicação OSI, tais como, de monitorização de configuração, de desempenho, de falhas, de segurança e de contabilização [40]. A monitorização de configuração tem como objetivo permitir ao utilizador a realização da monitorização de toda a estrutura física e lógica da rede, ou seja, o utilizador pode organizar e modificar os sistemas conforme a sua necessidade. Na monitorização de desempenho devem estar contidas as funcionalidades que permitem ao gestor da rede ter a capacidade de avaliar o comportamento da rede e dos vários dispositivos que a constituem, e ainda avaliar a eficiência das atividades de comunicação. Na monitorização de falhas devem estar incluídas funcionalidades que proporcionam ao utilizador a deteção, o isolamento e a correção de uma operação anormal. Na monitorização de segurança devem ser abrangidas funções necessárias a uma operação correta e para proteção dos objetos monitorizados. Por fim, na monitorização de contabilização o gestor tem a possibilidade de usufruir de funcionalidades que lhe permitem determinar o custo associado à utilização dos recursos da rede (quais e quanto desses recursos o sistema estão a ser utilizados) [3].

2.4. FERRAMENTAS DE MONITORIZAÇÃO

A existência de ferramentas capazes de proporcionar uma monitorização de uma rede com diversos equipamentos tem vindo a ser muito vantajosas para um gestor. Neste ponto são abordadas algumas ferramentas que exploram o SNMP para realizar a monitorização remota de equipamentos.

2.4.1. NAGIOS

Originalmente denominado de *Netsaint*, o Nagios foi criado por Ethan Galstad e a sua equipa de programadores encontra-se espalhada pelo mundo, dedicados a melhorar a ferramenta, no desenvolvimento de novos *plugins*, na correção de *bugs*, no melhoramento da interface Web, na produção e tradução da sua vasta documentação de suporte, entre outras atividades. Este *software* de monitorização de redes é distribuído livremente, através de uma licença *General Public Licence* (GPL). Apesar de ser projetado para redes de grande dimensão, o seu desempenho em redes pequenas é também excelente [16].

A utilidade do Nagios na monitorização de redes depende da sua expansão através de *plugins*, complementos escritos em *Common Gateway Interface* (CGI), ou noutra linguagem interpretada (*Partical Extration and Report Language* (PERL), *Hipertext Preprocessor* (PHP), Python, etc.), podendo ser desenvolvidos por diferentes programadores. Encontram-se disponíveis no sítio da Internet do *software*, vários *plugins* oficiais para a gestão [17].

O Nagios oferece várias funcionalidades, tendo como principais [16][18]:

- Monitorização de serviços de rede, como *Hypertext Transfer Protocol* (HTTP), *Post Office Protocol* (POP3), *Simple Mail Tranfer Protocol* (SMTP), telnet, entre outros;

- Notificação em caso de problemas em dispositivos de rede, através de correio eletrónico, SMS ou outro sistema de comunicação em tempo real;

- Facilidades dos utilizadores desenvolverem os seus próprios serviços de monitorização, através dos *plugins*;

- Interface Web para monitorização da rede, permitindo visualizar o estado dos vários dispositivos, o histórico de notificações de problemas, os *logs*, entre outros;

- Rotatividade automática de arquivos de *logs*.

O funcionamento de um sistema de monitorização Nagios assenta numa arquitetura gestor/agente (servidor/cliente). No sistema de monitorização, o servidor é o elemento principal, sendo nele que se encontram todas as informações, configurações e informações obtidas através da monitorização dos agentes do sistema. A informação dos dispositivos é constantemente atualizada pelo sistema Nagios, e de acordo com os seus resultados, executa as ações configuradas pelo utilizador. Uma dessas ações é o envio de um alerta (*email*, SMS, entre outros) para o utilizador de um problema ocorrido num dispositivo gerido.

A interface Web, Figura 6, proporciona ao utilizador uma grande variedade de informações, devidamente organizadas com os assuntos envolvidos. Utiliza um sistema de cores para demonstrar o estado do serviço (verde para normal, amarelo para uma situação de alerta e vermelho para situação critica ou de erro) diferindo de outras ferramentas de rede que mostram o tempo decorrido graficamente [16].

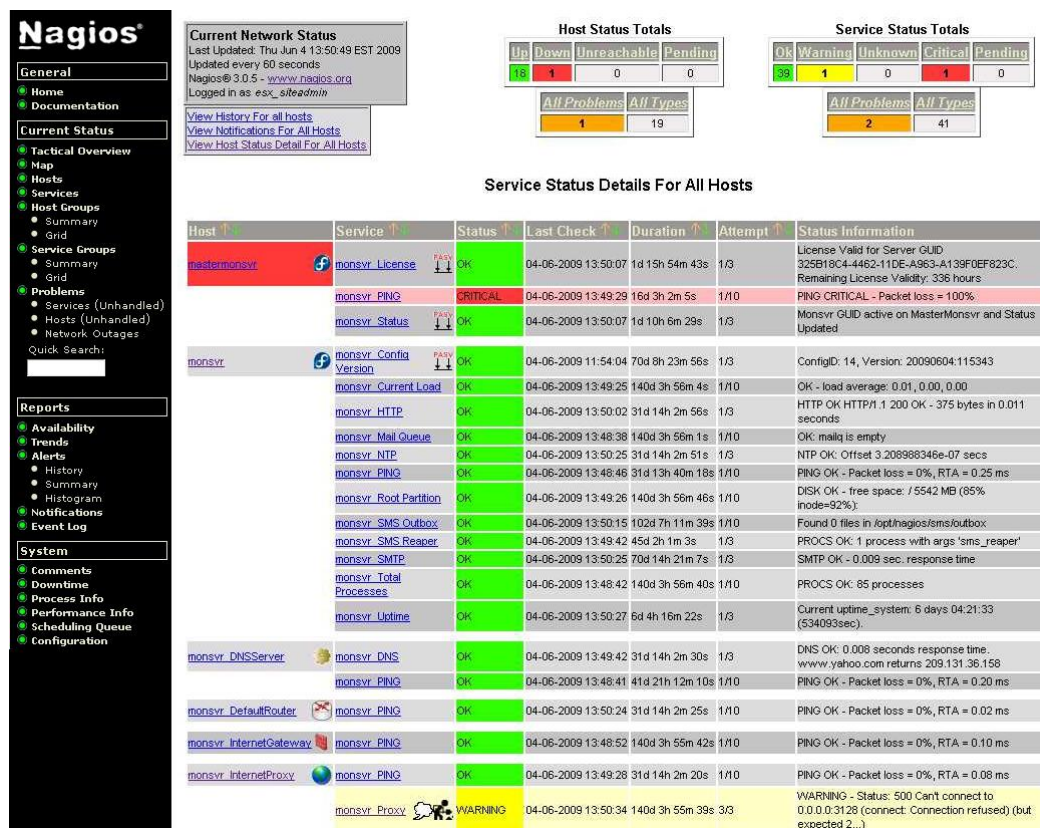



Figura 6 – Interface Nagios [19].

Através da interface Web é possível revelar acontecimentos de problemas, que ocorreram num intervalo de tempo selecionado, quem foi notificado, qual a situação do servidor ou serviço que estava a ser prejudicado nesse período de tempo (*logs*). É ainda possível através de um ambiente gráfico visualizar os dispositivos monitorizados e distingui-los consoante o sistema operativo utilizado e o tipo de dispositivo (servidor, router, entre outros), como pode-se observar na Figura 7.

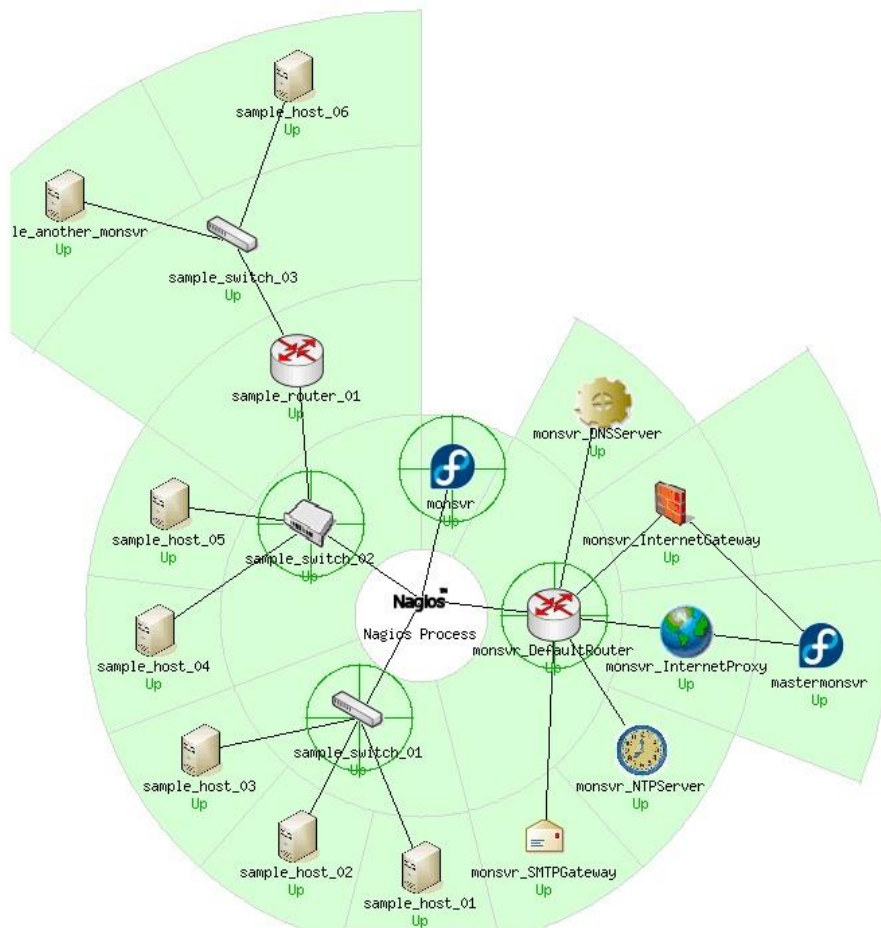



Figura 7 – Dispositivos monitorizados no Nagios [19].

A utilização do Nagios na monitorização de redes tem vindo aumentar ao longo do tempo em todo mundo. Este destaca-se por algumas características (já referidas anteriormente) que tornam o *software* numa excelente ferramenta, concorrendo até com algumas ferramentas comerciais existentes no mercado [16].

2.4.2. ZABBIX

O Zabbix é uma ferramenta de monitorização de redes desenvolvido por Alexei Vladishev, em 1998, projetado para uso privado num banco. Só posteriormente em 2001 é que foi disponibilizado com licença GPL [26]. Esta ferramenta foi criada para monitorar e controlar o estado de serviços de rede, servidores e outros dispositivos de rede em tempo real, tudo isto numa única ferramenta [20].

Esta ferramenta suporta a maioria de sistemas operativos (Windows, Unix, entre outros) e tem uma interface gráfica bastante simples para o utilizador, o que o torna numa das mais completas ferramentas de monitorização de redes [2]. Utiliza um mecanismo de notificações flexível, assente numa arquitetura cliente-servidor composta por um *software* agente (cliente) e um gestor (servidor) que permite realizar envio de dados do cliente para servidor sem que o servidor o tenha solicitado. Essas notificações para o administrador podem ser efetuadas através de correio eletrónico (*email*), *Short Message Service* (SMS) e alertas sonoros na interface Web [20].

As principais características do Zabbix são [2]:

- Autenticação e encriptação dos dados;
- Permissões flexíveis de utilizadores;
- Interface gráfica Web para visualização de recursos monitorizados;
- Notificação de eventos predefinidos por correio eletrónico;
- Alta eficiência dos agentes para as plataformas Unix e WIN32;
- Sistema de monitorização centralizado;
- Suporte para o SNMP (v1, v2 e v3).

O sistema de monitorização Zabbix é composto por um *software* agente (cliente) e software gestor (servidor). O servidor é o elemento principal do sistema de monitorização Zabbix, é nele que estão todas as informações, configurações e informações obtidas através da monitorização dos dispositivos de rede (agentes). É ainda o responsável por gerar os alertas a partir dos dados recebidos e notificar os administradores de rede, caso ocorra um

determinado evento (erro, problema, etc.) num dos componentes da rede. O agente do Zabbix é um *software* instalado no dispositivo gerido para monitorizar recursos locais, aplicações e arquivos de configuração ou de *logs*. Este envia ao servidor informações quando solicitado por esse, ou notifica-o caso ocorra alguma anomalia no sistema.

A interface Web do Zabbix proporciona o acompanhamento do desempenho da rede e dos dispositivos monitorizados (Figura 8), tais como, o seu estado, a carga do processador, o número de processos, a memória total ou a que esta ser utilizada, entre outras. Permite a criação de gráficos de tendências para que se possa visualizar o seu desempenho ao longo do tempo. Todas as configurações do sistema são efetuadas através da interface Web. Além disso, disponibiliza monitorização através de gráficos em tempo real, para que o administrador possa acompanhar o desempenho de um determinado dispositivo [2].

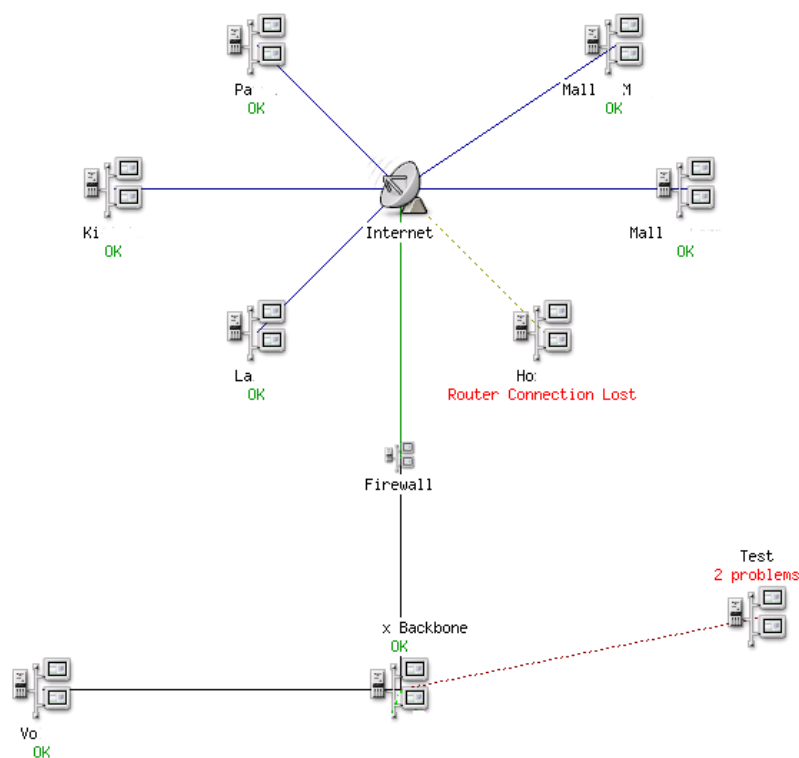



Figura 8 – Visualização da rede no Zabbix [28].

2.4.3. Cacti

O Cacti é uma ferramenta gráfica de monitorização de redes desenvolvida por IAn Berry, disponibilizando recursos bastantes avançados para serem utilizados em redes simples ou complexas. Desenvolvido em PHP o Cacti utiliza a estrutura do *Round Robin Database Tool* (RRDTool) para armazenamento de dados (base de dados MySQL) e criação de

gráficos. As informações necessárias para a criação de gráficos são obtidas através do protocolo SNMP, que periodicamente vai atualizando as informações dos dispositivos geridos [21].

Esta ferramenta disponibiliza ao utilizador uma interface intuitiva e de fácil utilização, podendo assim ser inteligível por utilizadores menos experientes bem como utilizadores com elevada experiência [23]. Proporciona as seguintes vantagens ao utilizador [22]:

- Simplicidade e rapidez de configuração de um dispositivo de rede;
- Interface flexível construída em PHP / MySQL;
- Armazenamento de dados ao logo de tempo (*logs*),
- Expansão da ferramenta através de *plugins*, como PHP *Network Weathermap* (onde se pode visualizar o mapa da rede gerida, bem como o estado dos dispositivos).

A arquitetura de funcionamento do Cacti é dividida em três diferentes tarefas, descritas na Figura 9.

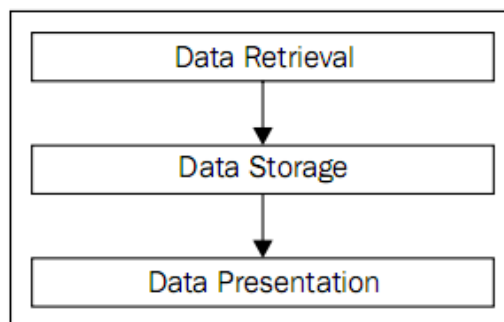



Figura 9 – Arquitetura do Cacti [22].

- **Data Retrieval**

A primeira tarefa do funcionamento consiste na recolha de dados. O Cacti utiliza a ferramenta Poller, executada no sistema operativo onde se encontra instalada. Atualmente, as infraestruturas de redes envolvem uma grande quantidade de diferentes dispositivos (servidores, *routers*, etc.). Para obter os dados dos dispositivos remotos, o Cacti utiliza principalmente o SNMP. Assim, todos esses dispositivos que utilizem o SNMP podem ser monitorizados através do Cacti [22].

- **Data Storage**

A segunda tarefa consiste no armazenamento de dados. O RRDTool é o responsável pelo armazenamento em base de dados dos dispositivos obtidos, através de SNMP, de uma forma compacta, que não se expandem ao longo do tempo. É por isso, que é rápida e fácil a criação de gráficos [22].

- **Data Presentation**

Por último, a tarefa de apresentação dos dados, que anteriormente foram obtidos e armazenados. O RRDTool tem a capacidade de gerar gráficos, função que valoriza, e muito. Para além disso gera também itens de ajuda na compreensão de cada gráfico, tais como legendas, máximos, mínimos, etc. Essa funcionalidade é bastante importante e útil para o utilizador, devido ao facto dos gráficos serem consultados através de uma interface Web [22]. Na Figura 10 pode-se visualizar como essa informação é apresentada na interface.

Figura 10 – Gráficos do Cacti [22].

A interface Web do Cacti proporciona uma interação simples ao utilizador, como se pode visualizar na Figura 11 e onde, do lado esquerdo pode observar-se o menu de utilização da aplicação. Na parte superior existem menus que podem alternar entre o modo de visualização gráfica (ilustrado na Figura 10) e o menu de opções de configuração (ilustrado na Figura 11).

Figura 11 – Interface do Cacti [22].

3. ESPECIFICAÇÃO DO GATEKEEPER E PLATAFORMA DE DESENVOLVIMENTO

O sistema GateKeeper, nome atribuído pela NextToYou para a aplicação a desenvolver no âmbito desta tese, exige determinados requisitos funcionais para o seu desenvolvimento e monitorização de GateBoxes.

A necessidade de monitorização de um conjunto de GateBoxes pode ser equiparada à de empresas de *step-to-box*, com dispositivos distribuídos por uma dada área geográfica e ainda a empresas de grande dimensão, com um elevado conjunto de equipamentos (*routers*, *switches*, impressoras, entre outros). Essa monitorização torna-se indispensável, principalmente em termos de resolução de problemas que possam existir nos serviços proporcionados. No caso de redes de GateBoxes, com o número elevado de funcionalidades que integram (descritas anteriormente no ponto 1.1), é essencial que estas sejam permanentemente monitorizadas, para que se possam prevenir casos de falhas do

funcionamento do sistema ou realizar notificações aquando da deteção de problemas, para uma rápida resolução.

Durante este capítulo serão explicados com mais detalhe os requisitos funcionais necessários para a criação do sistema GateKeeper, e ainda será apresentada a plataforma de desenvolvimento utilizada no sistema.

3.1. Requisitos Funcionais

As ferramentas descritas no ponto 2.4 permitem um conjunto de parâmetros na monitorização de dispositivos. Essas ferramentas podiam ser utilizadas para monitorizar GateBoxes. Contudo, a NextToYou pretende desenvolver uma plataforma para esse efeito, com funcionalidades que proporcionam uma monitorização mais eficaz e flexível. Algumas funcionalidades dessas ferramentas foram igualmente definidas como necessárias na aplicação Web a desenvolver. Uma dessas funcionalidades é a capacidade de oferecer ao utilizador uma interface simples e intuitiva. O Nagios integra alguns aspetos interessantes para o desenvolvimento da plataforma de monitorização, tais como: a utilização de um sistema de cores para demonstrar o estado do serviço do sistema (verde para normal, amarelo para uma situação de alerta e vermelho para uma situação crítica ou erro), e a visualização do estado da ligação dos dispositivos monitorizados (*Up*/*Down*) em tempo real. Este último aspeto, também é proporcionado pela ferramenta Zabbix.

Dos protocolos de gestão descritos no ponto 2.3, o que será utilizado na monitorização é o SNMP. Este, também foi o sugerido pela NextToYou, por já se encontrar instalado nas GateBoxes.

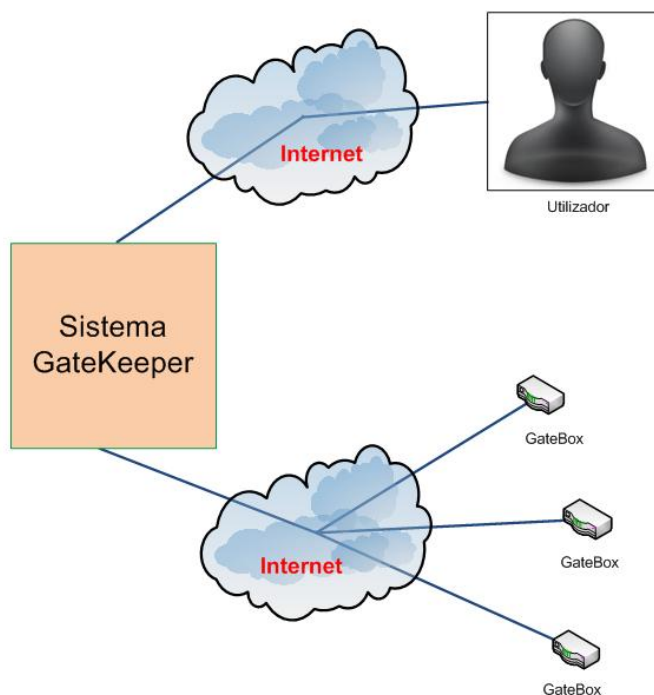



Figura 12 – Arquitetura de utilização do sistema GateKeeper.

Um dos requisitos é que o sistema deverá proporcionar ao utilizador uma interação através de uma aplicação Web. Na interface dessa aplicação deverá ser possível identificar os dispositivos a monitorizar bem como visualizar a sua informação (características, informações obtidas pelo protocolo SNMP, o estado da ligação, entre outras). Também deverá permitir a consulta e visualização ao utilizador de todas as notificações enviadas pelas GateBoxes. Além disso, deverá conter uma área de configuração dos parâmetros necessários para a monitorização e notificação SNMP (perfil utilizado pelas GateBox, *email* de notificação, entre outros). Por último, a definição de OIDs privados que vai permitir uma monitorização mais específica a determinados objetos das MIBs das GateBoxes. Para cada um destes, a interface deve oferecer diferentes funcionalidades para tornar o sistema flexível, tais como: consulta, inserção, modificação e eliminação. Na aplicação Web deve-se privilegiar os requisitos funcionais em vez do aspeto visual da interface gráfica.

Outro requisito que deverá ser garantido pelo sistema GateKeeper é a sua independência da aplicação Web, isto é, não sendo necessário que a aplicação Web esteja ativa para que se realize a monitorização. Para isso, no servidor do sistema é necessária que existam alguns componentes (*scripts*, *daemons*, entre outros), que assegurem a monitorização de dispositivos remotos. Estes componentes deverão verificar o estado das ligações com as

GateBoxes, a obtenção e armazenamento de informação através do protocolo SNMP, a capacidade de receção de notificações provenientes dos dispositivos monitorizados e o envio de alertas para o utilizador (gestor do sistema) no caso de receção de uma notificação.

Adicionalmente, o GateKeeper deverá integrar uma base de dados onde será armazenada toda a informação definida na interface Web, bem como a informação obtida na execução de determinados componentes (*scripts*) do sistema.

Para desenvolver o sistema GateKeeper é pretendida uma estreita integração com a *framework* Web que suporta as restantes aplicações NextToYou já em funcionamento nos seus equipamentos, sendo sugerida e fornecida pela NextToYou uma plataforma de desenvolvimento criada na empresa, a WebForge (plataforma descrita no ponto 3.2). Esta plataforma opera em sistemas operativos Linux, sendo que a distribuição aconselhada pela empresa foi o Ubuntu [25].

3.2. PLATAFORMA DE DESENVOLVIMENTO

A plataforma de desenvolvimento para o trabalho proposto foi fornecida pela NextToYou, e denomina-se de WebForge. Trata-se de uma *framework* em PHP 5.3 [45], que foi criada com o intuito de facilitar o desenvolvimento de aplicações Web para a integração com sistema da empresa.

O seu desenvolvimento tem como objetivo o fornecimento ao programador de uma base fácil de criação de aplicações Web através da disponibilização de bibliotecas NextToYou e bibliotecas *open source* (PHP, JavaScript), reconhecidas pela sua qualidade. Contém ainda, um conjunto de outros recursos, como ícones e *Cascade Style Sheet* (CSS), que o programador pode utilizar no seu desenvolvimento. Além disso, possibilita a implementação do modelo *Model-View-Controller* (MVC). O modelo MVC é uma arquitetura ou padrão que permite dividir as funcionalidades da aplicação Web em camadas (*Model, View e Controller*) [51]. Uma das vantagens é o controlo distinto da programação lógica da aplicação e da apresentação gráfica da interface [48].

3.2.1. ORGANIZAÇÃO DE UM PROJECTO

A Figura 13 representa a estrutura de pastas de um projeto WebForge após a sua geração e instalação. Na geração de um projeto definem-se o nome do projeto (*myproject*) e o autor, entre outros dados. Alguns destes dados são utilizados para a criação de vários ficheiros e pastas na estrutura do projeto, por exemplo `myproject.log` e dentro a pasta `NextToYou` uma pasta `MyProject`. Na criação de um determinado projeto é necessário a instalação e configuração de uma base de dados, para interagir com a aplicação Web. O ficheiro (*`webforge.sql`*) para criação da base de dados é fornecido na pasta de instalação da ferramenta. De salientar que todo o processo de gestão de sessões (*cookies*) da aplicação já se encontra desenvolvido na aplicação base do projeto.

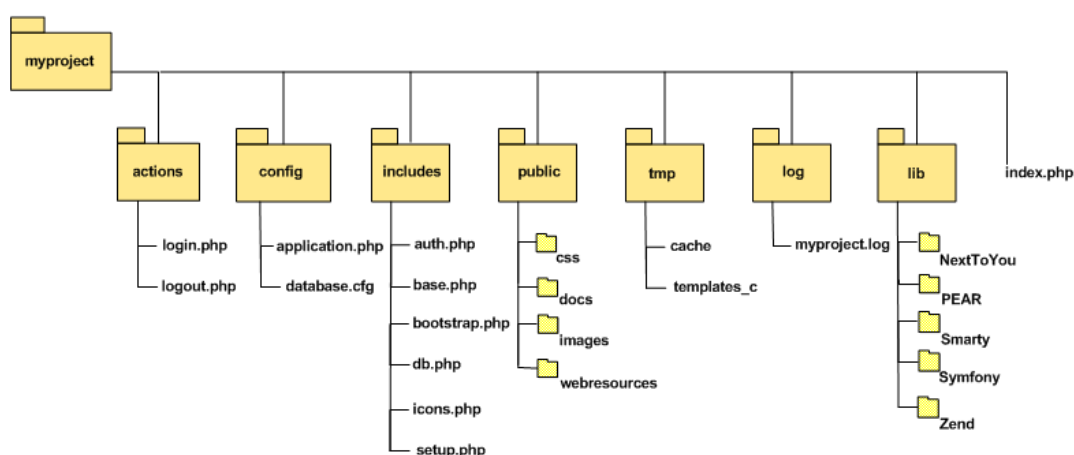



Figura 13 – Organização de um projecto na WebForge.

Como se pode observar na Figura 13, o único ficheiro que se encontra na raiz do projeto é o `index.php`. Contudo podem ser adicionados mais ficheiros, como o `GateBox.php` para apresentar páginas relativas ao perfil de cada GateBox, de consulta e de inserção, entre outros. Estes ficheiros denominam-se de controladores, por desempenharem funcionalidades de tratamento e reencaminhamento de informações na interface Web.

Na pasta *`actions`* é onde o utilizador deve colocar os scripts em PHP que executam determinadas ações como por exemplo, inserir, modificar, eliminar GateBoxes, etc. No projeto base já se incluem os ficheiros `login.php` e `logout.php`. No primeiro ficheiro é efetuada a verificação dos dados inseridos nos campos de autenticação da aplicação, e caso a informação inserida esteja correta, é então iniciada uma nova sessão

para aquele utilizador, reencaminhando-o para a página inicial da aplicação. O segundo ficheiro realiza o encerramento da sessão.

Os ficheiros que fazem parte da configuração da aplicação encontram-se na pasta `config`. O ficheiro `application.php` é o ficheiro nuclear de configuração da aplicação. É nele que se encontram as informações que foram inseridas aquando da instalação e criação do projeto. Um outro ficheiro que se encontra nesta secção é o `database.cfg` que contém as informações necessárias para o acesso à base de dados.

Na pasta `includes` encontram-se os *scripts* PHP que são incluídos nos controladores (ficheiros da raiz do sistema, por exemplo `index.php`) que preparam todos os elementos necessários para o pedido efetuado (*request*). Esses ficheiros desempenham determinadas funções, tais como, a inicialização da aplicação (`bootstrap.php`), a leitura das configurações (`database.cfg`) necessárias para o acesso à base de dados (`db.php`), a verificação da sessão atual de um utilizador (`auh.php`), a invocação de serviços (`setup.php`), assim como o *logger* e a instância Smarty (*template* base da aplicação). Por fim, o ficheiro `base.php` efetua o tratamento dos elementos dos ficheiros da pasta `includes` e envia os objetos para o *template*.

No diretório `public` encontram-se recursos estáticos utilizados na aplicação, como por exemplo imagens, CSS, JavaScript (pasta `webresources`). Além disso, contém uma pasta docs, com um tutorial em HTML da ferramenta de desenvolvimento WebForge.

Os ficheiros temporários gerados pelo *template* Smarty são colocados na pasta `tmp` que contém duas pastas, a `cache` e a `templates_c`, a primeira para a cache gerada pelo template e a outra para templates compilados, respetivamente. O *log* criado pela aplicação (`myproject.log`) é colocado numa pasta com o nome de `log`.

Por último, o diretório `lib` é onde se colocam as bibliotecas utilizadas pela ferramenta de desenvolvimento WebForge, desde a de geração de páginas, à de acesso às bases de dados, etc. A biblioteca `NextToYou` contém o seu elemento principal `WebCore` (diretório), que permitirá ao utilizador o desenvolvimento de classes PHP para a aplicação. Dentro da pasta `NextToYou` é criada uma pasta com o nome do projeto onde o programador deve colocar todas as classes que constituem a aplicação, as classes de interação com a base de dados, bem como os ficheiros HTML que foram utilizados na aplicação. Além disso, a WebForge

utiliza bibliotecas de utilização livre já desenvolvidas, como as bibliotecas PEAR (PHP *Extension and Application Repository*), o Smarty, o Symfony e o Zend, descritas posteriormente neste capítulo. Estas encontram-se em pastas separadas dentro do diretório `lib`.

3.2.2. PEAR

O PEAR é uma plataforma e um sistema de distribuição de componentes em PHP, que tem como objetivo fornecer [29]:

- Uma biblioteca estruturada de código aberto em PHP para os utilizadores;
- Um sistema para a distribuição de código e de gestão de pacotes;
- Um padrão para o desenvolvimento de código PHP;
- Uma biblioteca com extensões PHP, a PHP *Extension Community Library* (PECL);

O código no PEAR é dividido em pacotes que proporcionam ao utilizador algumas funções, como autenticação, controlo de erros, *caching*, acesso a base de dados, criptografia, entre outros. A WebForge utiliza o PEAR para a interação com a base de dados através do módulo `PEAR/DB.php`.

3.2.3. SMARTY

O Smarty é uma biblioteca (*template*) para PHP que permite o controlo distinto da programação lógica da aplicação e da apresentação gráfica da interface [48]. Sendo útil, no caso de um *designer* gráfico não ser um programador da aplicação e pretender alterar algo na parte gráfica do utilizador, não necessita de efetuar alterações na programação lógica, e vice-versa. O Smarty utiliza *tags* para relacionar e substituir dados entre os ficheiros [30]. Na Figura 14 pode-se visualizar o princípio de funcionamento do Smarty. Na ferramenta WebForge, este *template* é utilizado para se fazer a separação do código PHP do HTML.

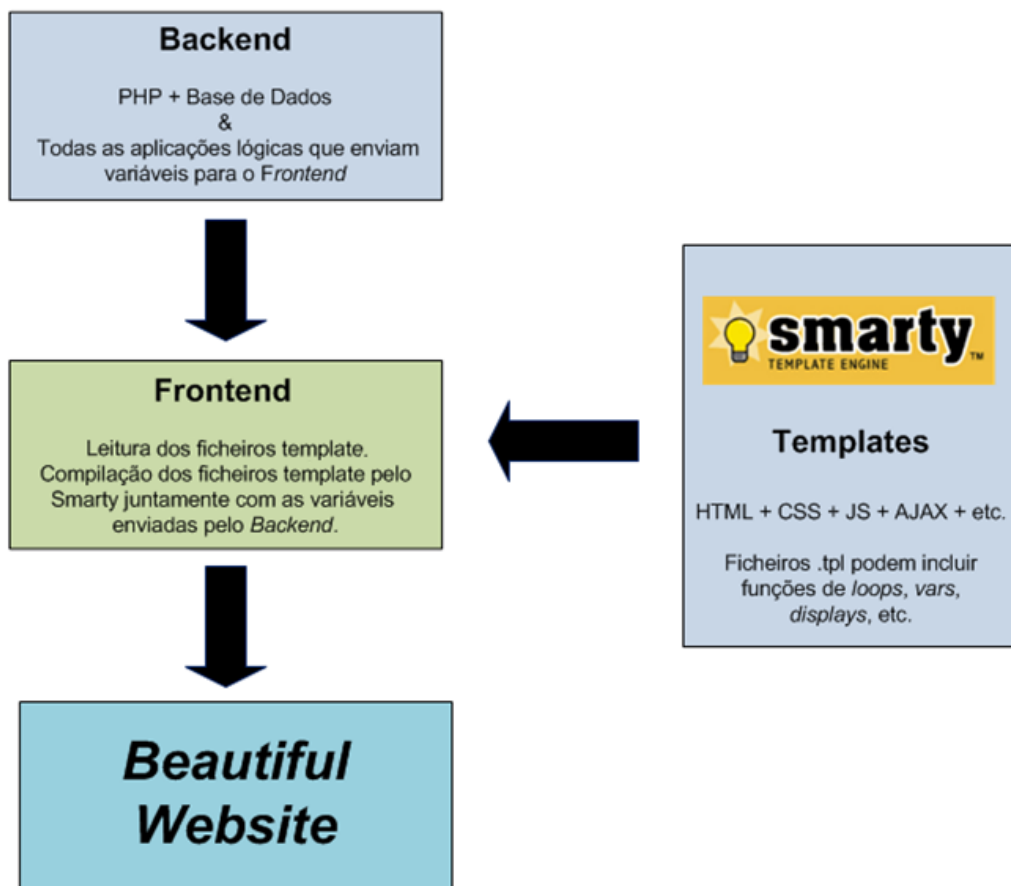



Figura 14 – Principio de funcionamento do Smarty [49].

3.2.4. SYMFONY

A Synfony é uma biblioteca de classes escrita em PHP que fornece uma arquitetura, módulos e ferramentas para desenvolver aplicações Web. O número reduzido de pré-requisitos para a sua instalação e configuração faz com que seja muito utilizada, necessitando apenas de um servidor Web com o interpretador PHP instalado. A Symfony tem como objetivo a construção de aplicações robustas num contexto empresarial, dando aos utilizadores um controlo total sobre a configuração, desde os diretórios às bibliotecas externas. Numa aplicação Web a integração de módulos para realização de determinadas tarefas (testar, depurar, documentar o projeto) pode desempenhar um papel importante na sua criação [31].

Na ferramenta WebForge são utilizados módulos do Symfony, sendo um desses, o *Dependency Injection Container* [50] (que cria instâncias num objeto), que é um constituinte do sistema de serviços implementado na ferramenta de desenvolvimento.

3.2.5. ZEND FRAMEWORK

O Zend é uma *framework* PHP para desenvolver aplicações Web, com programação orientada a objetos e serviços em PHP. Os módulos da estrutura da *framework* são únicos e cada componente é projetado para ser independente dos outros ou ter poucas dependências [32].

A WebForge utiliza vários módulos do Zend Framework, o mais relevante é o Zend_Log, módulo de criação de registos. Este módulo utiliza uma classe estática que tem como objetivo formatar e filtrar as mensagens enviadas para o ficheiro de registo (*log*) da aplicação.

4. GATEKEEPER

O sistema GateKeeper desenvolvido é uma ferramenta de monitorização que interage com o utilizador através de uma interface Web. O objetivo principal desta ferramenta é possibilitar a monitorização remota do funcionamento das GateBoxes e também de registar e guardar avisos e alarmes gerados por elas.

Na Figura 15 ilustram-se todos os componentes da arquitetura do sistema GateKeeper. O utilizador interage com o sistema através de uma interface Web que foi desenvolvida na plataforma WebForge (ponto (1) da figura). Toda a informação inserida, consultada, alterada e eliminada, que se realiza na interface resulta em operações com a base de dados. Essas operações são efetuadas através de classes em PHP 5.3 [45] (2). Contudo, nem toda a informação contida na base de dados é inserida pela interface Web, podendo ser resultantes de operações de inserção realizadas pelos *scripts* (3) que desempenham tarefas como por exemplo a leitura de informação das GateBoxes através do protocolo SNMP, a verificação do estado desses dispositivos e ainda a análise de notificações geradas por eles. Adicionalmente este *script* pode também enviar alertas previamente configurados, para o utilizador/gestor através de correio eletrónico (4). Para que todos *scripts* sejam executados periodicamente foi utilizado o *crontab* (5). Além deste, recorreu-se a um *daemon,* o *snmptrapd,* que torna possível a receção de notificações geradas pelas GateBoxes

registando-as num ficheiro (6). As GateBoxes a monitorizar possuem um *daemon* (*snmpd*) ativo para troca de informação com o sistema GateKeeper (7).

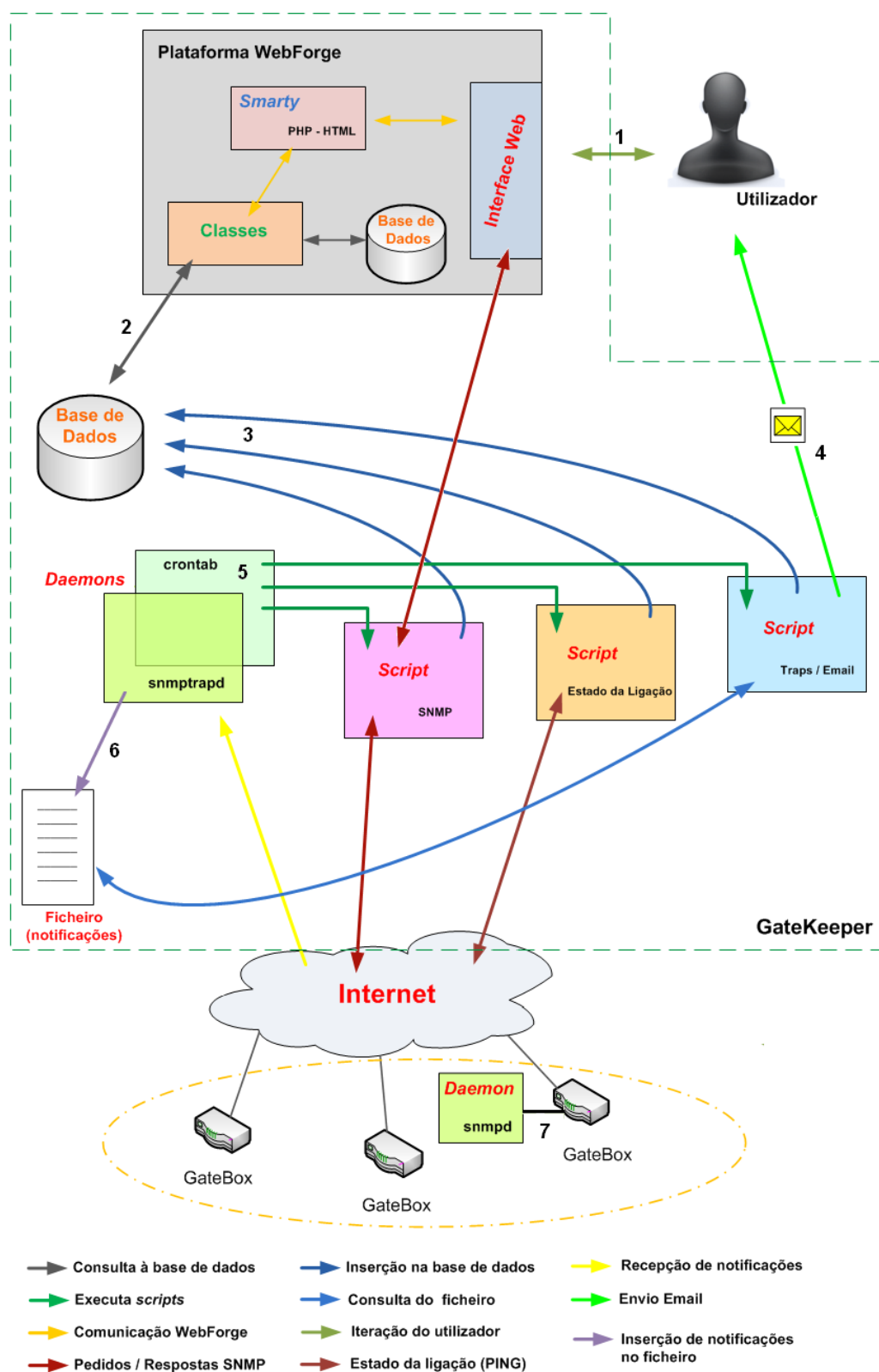



Figura 15 – Arquitetura do sistema GateKeeper.

Ao longo deste capítulo será descrito todo o desenvolvimento efetuado para cada elemento do sistema GateKeeper. Como primeiro ponto, será apresentada uma descrição da base de dados que foi criada para armazenamento de dados do sistema, as tabelas que a constituem, e as relações entre estas. De seguida será focado a utilização do *template* Smarty e das classes para comunicação com a base de dados. Posteriormente, a arquitetura da interface Web bem como as suas funcionalidades. E por fim, as particularidades dos *daemons* e dos *scripts* desenvolvidos para suporte do sistema.

4.1. BASE DE DADOS

A base de dados é um elemento central no sistema de monitorização de GateBoxes, pois é nela que será armazenada toda a informação sobre o sistema, a informação recolhida através do protocolo SNMP, bem como a informação configurada pelo utilizador através da interface Web. Foram definidas várias tabelas de modo a acomodar os diversos dados necessários, assim como definidas relações entre tabelas para uma maior eficácia. Na criação da base de dados utilizou-se uma ferramenta em PHP, o phpMyAdmin. Nesta ferramenta, podem-se realizar várias operações de gestão da base de dados, tais como a criação, alteração e eliminação de bases de dados. Dentro destas é possível efetuar a inserção, a remoção, a alteração de tabelas e campos destas, assim como a definição de relações nas tabelas, entre outras configurações. Pode-se visualizar na Figura 16 a estrutura da base de dados criada para a aplicação, bem como as respetivas relações entre as tabelas que a constituem.

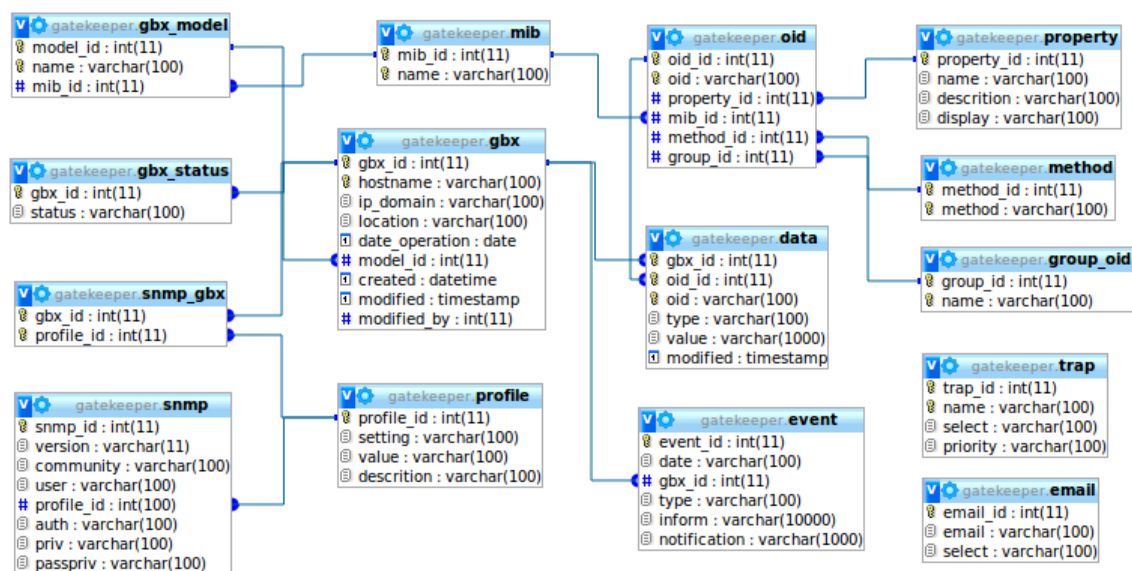



Figura 16 – Tabelas da base de dados desenvolvida.

A tabela gbx contém os dados da GateBox como, o *hostname* para identificação, o *IP/domain* necessário para comunicação, a morada e a data de início de funcionamento. Foram criadas outras tabelas, como a gbx_model que armazena o modelo, a tabela gbx_status, de armazenamento do estado da ligação, a tabela data onde será armazenada toda a informação obtida através do protocolo SNMP para cada uma das GateBoxes e a tabela event de armazenamento de notificações enviadas por cada dispositivo. Relacionada ainda com a tabela gbx, está também a tabela snmp_gbx de definição do perfil a ser utilizado no protocolo SNMP. Esta tabela pertence a um grupo de tabelas (snmp, snmp_gbx e profile) de armazenamento de informações (versão, comunidade, autenticação, encriptação, entre outros) acerca dos perfis utilizados no SNMP para cada uma das GateBox, pertencendo cada a uma categoria (gbx_model) que contém uma MIB associada (mib). A tabela mib está associada com as tabelas oid, property, method e group_oid, tabelas que armazenam informação sobre OIDs. Por último, é utilizado um grupo de tabelas (event, trap e email) para armazenamento de dados das notificações, configuração dos tipos de notificação, e envio de *emails*.

4.2. SMARTY E CLASSES

A interface Web desenvolvida foi criada através da plataforma WebForge, descrita no capítulo 3.2. Este desenvolvimento seguiu determinadas requisitos operacionais da plataforma, tais como, a utilização do *template* Smarty [48] para a separação do PHP do HTML na programação, a criação de classes para cada tabela da base de dados, entre outros.

A utilização do *template* Smarty permite a separar a programação lógica da apresentação gráfica (PHP do HTML). Para tal, nos ficheiros PHP devemos utilizar as designações específicas do *template* para a utilização deste. No seguinte excerto de código pode visualizar-se um exemplo da transferência de dados de PHP para HTML.

```
…
$boxes = Box::getBox($gbx_id);
$_tpl->assign('boxes', $boxes);
$_tpl->assign('loc', $loc);
$_tpl->display('box/listmap.html');
…
```

Os dados obtidos através do método getBox($gbx_id) da classe Box são armazenados na variável $boxes. O Smarty vai definir uma variável ('boxes') para ser utilizada no ficheiro HTML, esta definição é efetuada pelo método assign da classe _tpl (classe já existente na plataforma WebForge). Seguidamente, o método display origina a página HTML ('box/listmap.html').

No seguinte exemplo pode-se ver como é utilizada a informação através do *template* no HTML. Todas as variáveis definidas pelo Smarty estão contidas entre chavetas {}. Os valores das variáveis ($loc e $boxes) são atribuídos no código PHP para ser utilizado em HTML. Neste caso, a variável {$loc} contém a morada de uma GateBox. Em seguida, é construída uma tabela através da função foreach, que vai efetuar um ciclo para todos os elementos do *array* $boxes. Em cada um dos ciclos, é originado uma linha com várias colunas, as quais serão preenchidas com características do elemento do *array*.

```
…
<h3>
Local: {$loc}
</h3>
…
<table>
{foreach $boxes item=box}
<tr>
  <td>{$box->gbx_id}</td>
  <td>{$box->hostname}</td>
  <td>{$box->ip_domain}</td>
  <td>{$box->location}</a></td>
  </tr>{/foreach}
</table>
…
```

Desta forma, todos os parâmetros/valores provenientes dos ficheiros PHP são sempre transferidos para o HTML usando o *template* Smarty.

A comunicação com a base de dados é efetuada através de classes. Foi criada uma classe para cada uma das tabelas da base de dados, para realização de operações. Cada classe deve possuir determinados métodos (getId e getName) para que se possa utilizar o modelo especificado pela plataforma de desenvolvimento para interligação com a base de dados.

```
…
class BoxModel extends BaseModel
{
  const TABLENAME = 'gbx_model';
  const PRIMARY_KEY = 'model_id';

  public $model_id;
  public $name;
  public $mib_id;


  public function getId()
  {
       return $this->model_id;
  }

  public function getName()
  {
       return $this->name;
  }
…
```

A classe `BoxModel`, assim como todas as outras classes criadas para a comunicação com a base de dados, são uma extensão da classe `BaseModel`. A classe `BaseModel` é onde o programador define uma constante (`DB_HANDLER_ID`) para o nome da base de dados que pretende utilizar. Neste excerto, definem-se de constantes com o nome da tabela (`TABLENAME`) e a chave primária (`PRIMARY_KEY`), bem como variáveis, que posteriormente serão utilizadas nos métodos constituintes desta classe. A definição dessas constantes proporciona uma maior flexibilidade no código, como por exemplo, na alteração do nome da tabela na base de dados, o programador só altera o nome da tabela na classe uma vez. Para cada tabela são definidos métodos para realizar diferentes operações MySQL com a base de dados, tais como: `SELECT`, `INSERT INTO`, `DELETE`, `UDPDATE`, entre outros.

No seguinte extrato de código, pode observar-se um método para consulta de informações de uma determinada tabela na base de dados. Inicialmente é efetuada a ligação à base de dados (`DB_HANDLER_ID`) através do método `getHandler` da classe `Database`. Esta classe efetua a autenticação do utilizador com a base de dados, através dos dados fornecidos aquando da instalação e a criação do projeto na plataforma, e guarda-os no ficheiro `database.cfg`. Os dados que foram consultados com a operação `SELECT` são armazenados num vetor `$models` que vai ser retornado quando este método for solicitado.

```
public static function getAll()
```

```
{
 $db =
Database::getHandler(static::DB_HANDLER_ID);

 $query = sprintf("SELECT * FROM `%s`",
self::TABLENAME);

 $result = $db->query($query);

 $models = array();

 if (\PEAR::isError($result))
 {
  die($result->getDebugInfo());
 }

  while ($row = $result->fetchRow())
 {
  $models[$row[self::PRIMARY_KEY]] = new
static($row);
      }
  return $models;
}
```

Na Figura 17 ilustra-se a organização dos ficheiros desenvolvidos e pastas para a implementação da aplicação. No diretório principal do projeto foram criados os ficheiros PHP denominados por controladores (`index.php`, `gbx.php`, etc), cada um destes ficheiros corresponde na interface a um menu. A pasta `Model` é onde se encontram as classes PHP responsáveis pela interação com a base de dados e na pasta `Resources` encontram-se todos os ficheiros HTML desenvolvidos para a aplicação desenvolvida.

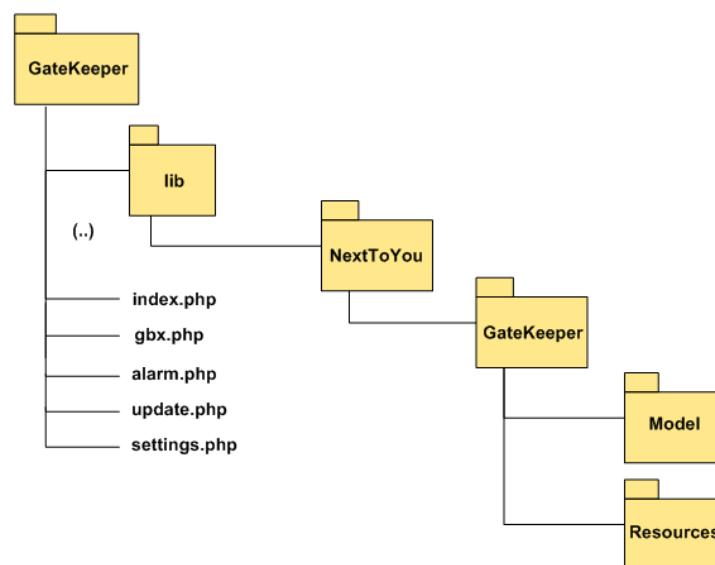



Figura 17 – Organização dos ficheiros e pastas desenvolvidos.

4.3. INTERFACE WEB

A interface Web da aplicação desenvolvida deve permitir ao utilizador (gestor do sistema) desempenhar determinadas tarefas, como por exemplo consultar, inserir, modificar e eliminar informações sobre as GateBoxes, alarmes, entre outras. Na Figura 18 ilustra-se os vários menus da aplicação necessários para monitorização de sistemas por parte do utilizador.

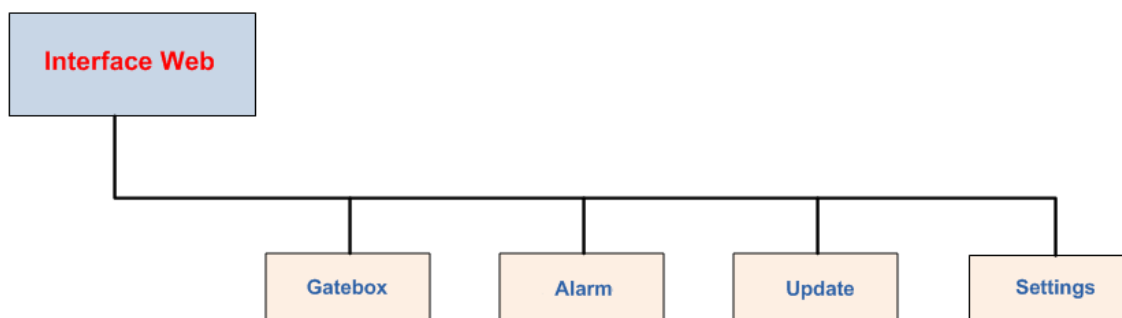



Figura 18 – Especificação da interface Web.

O menu GateBox permite que utilizador consulte, insira, modifique e apague informações das GateBoxes. Na inserção de uma nova GateBox o utilizador deve definir informações como o *hostname*, o IP/*domain*, o local/morada e a versão do SNMP, dados esses utilizados pelos *scripts* na monitorização do sistema. Ainda nesta secção o utilizador poderá modificar e apagar essas informações. A consulta de informação das GateBoxes é efetuada através da listagem de todas GateBoxes armazenadas na base de dados. Nesta consulta pode-se visualizar a informação inserida e obtida da GateBox através do protocolo SNMP, caso essa já se encontre armazenada na base de dados.

No menu Alarm é possível consultar as notificações geradas pelas GateBoxes. Essas são armazenadas na base de dados através de um *script*. Este por sua vez também informa o utilizador da existência de um evento através de correio eletrónico (*email*).

As informações obtidas através do protocolo SNMP podem ser atualizadas no menu Update da aplicação, caso o utilizador necessite, possibilitando ainda que a tarefa seja realizada para todas as GateBox existentes, ou apenas para uma específica.

Por último no menu Settings, o utilizador pode definir os parâmetros necessários para a monitorização dos dispositivos. A Figura 19 ilustra como se encontra organizado este menu.

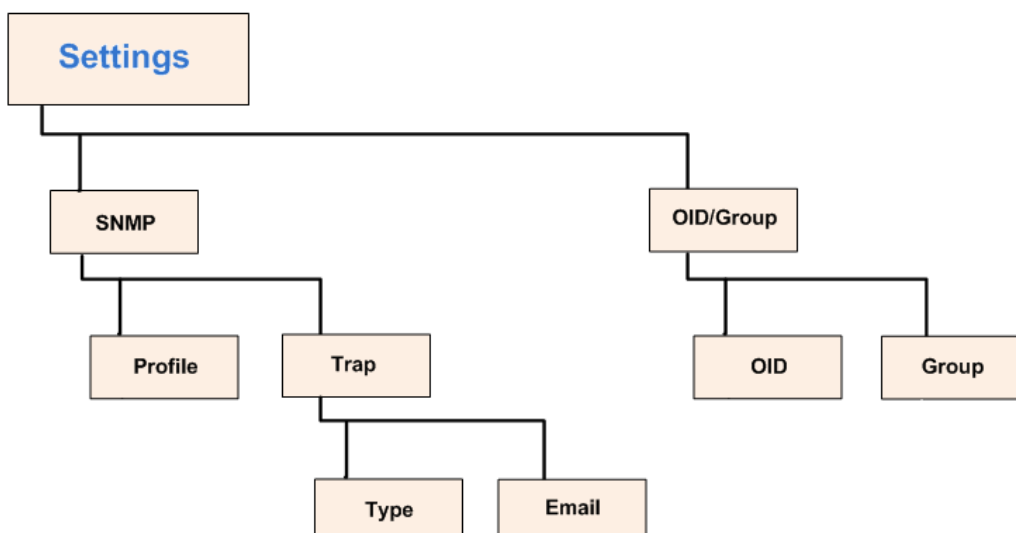



Figura 19 – Menu Settings.

O protocolo SNMP necessita de vários parâmetros para a realização dos pedidos, como por exemplo: a versão utilizada e a necessidade de *password*, ou não, no caso de utilizar uma versão com autenticação. O utilizador pode escolher, através de uma listagem, a versão, a autenticação e encriptação caso seja preferida, e também definir a *password* caso seja necessário. Tudo isto é efetuado numa subsecção da secção SNMP denominada por Profile. Para além da Profile, existe uma subsecção Trap que permite ao utilizador a definição, através da inserção, modificação e eliminação de tipos de notificações. Ainda nesta subsecção, pode ser feita a configuração do endereço de correio eletrónico necessário para envio de um alerta ao utilizador.

Na secção OID/Group existe uma subsecção OID, que desempenha um papel importante na aplicação e que faz com que a aplicação desenvolvida se diferencie de outras ferramentas já existentes de monitorização de dispositivos de rede. É nesta subsecção que o utilizador pode efetuar a consulta, inserção, modificação e eliminação de OIDs. Assim, permiti-se ao utilizador definir as informações que entenda ser mais relevantes na monitorização de GateBoxes. Para organizar os OIDs para visualização na interface Web foi criada uma subsecção Group. Nesta, o utilizador pode definir grupos para cada um dos OIDs de modo a simplificar a consulta da informação. Tanto para os OIDs como para os grupos pode-se efetuar a consulta, a inserção, a modificação e a eliminação dos mesmos.

Na Figura 20 pode-se observar a arquitetura da interface Web do sistema GateKeeper. Esta encontra-se dividida em menus principais: Home, GateBox, Alarm, Update e Settings.

Cada um destes menus vai ser descrito nesta secção com figuras ilustrativas das suas funcionalidades. A interface Web foi desenvolvida em linguagem inglesa por determinação da empresa NextToYou, sendo possível a apresentação multilíngue dos seus conteúdos através da *framework* WebForge, não sendo no entanto um requisito para o trabalho desenvolvido.

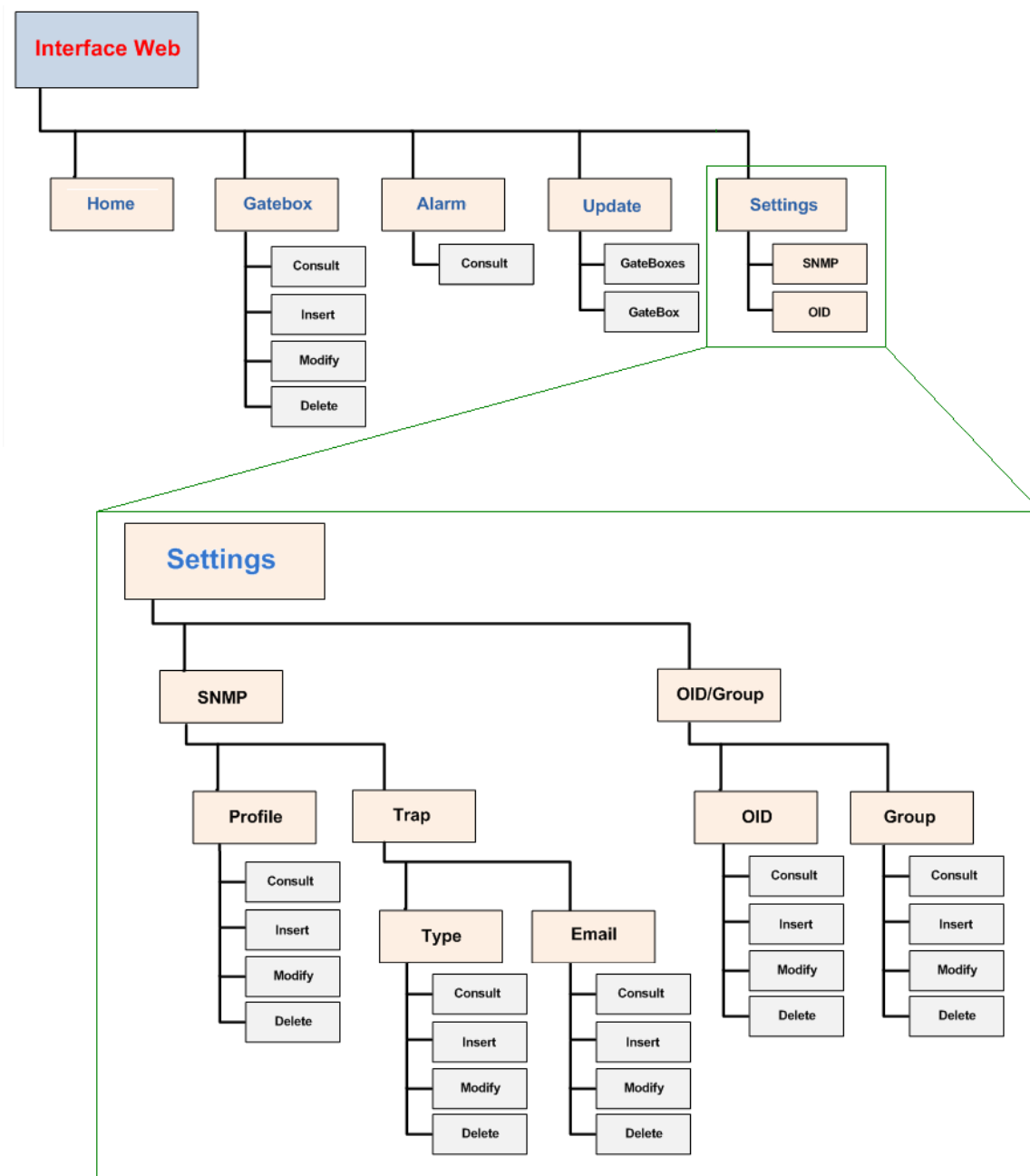



Figura 20 – Arquitetura do *software* desenvolvido.

Para iniciar a aplicação Web o utilizador deve efetuar a autenticação com *username* e *password*. Esta autenticação é obrigatória para qualquer desenvolvimento efetuado na WebForge, sendo utilizado para isso uma base de dados diferente da usada na aplicação GateKeeper. Essa base de dados é fornecida com a ferramenta de desenvolvimento apenas para ser utilizada na autenticação deste projeto conforme o pretendido pela NextToYou.

4.3.1. MENU HOME

O menu Home, ilustrado na Figura 21, é a página de apresentação da aplicação Web desenvolvida, onde se resume de uma forma sucinta as funcionalidades dos outros menus principais.

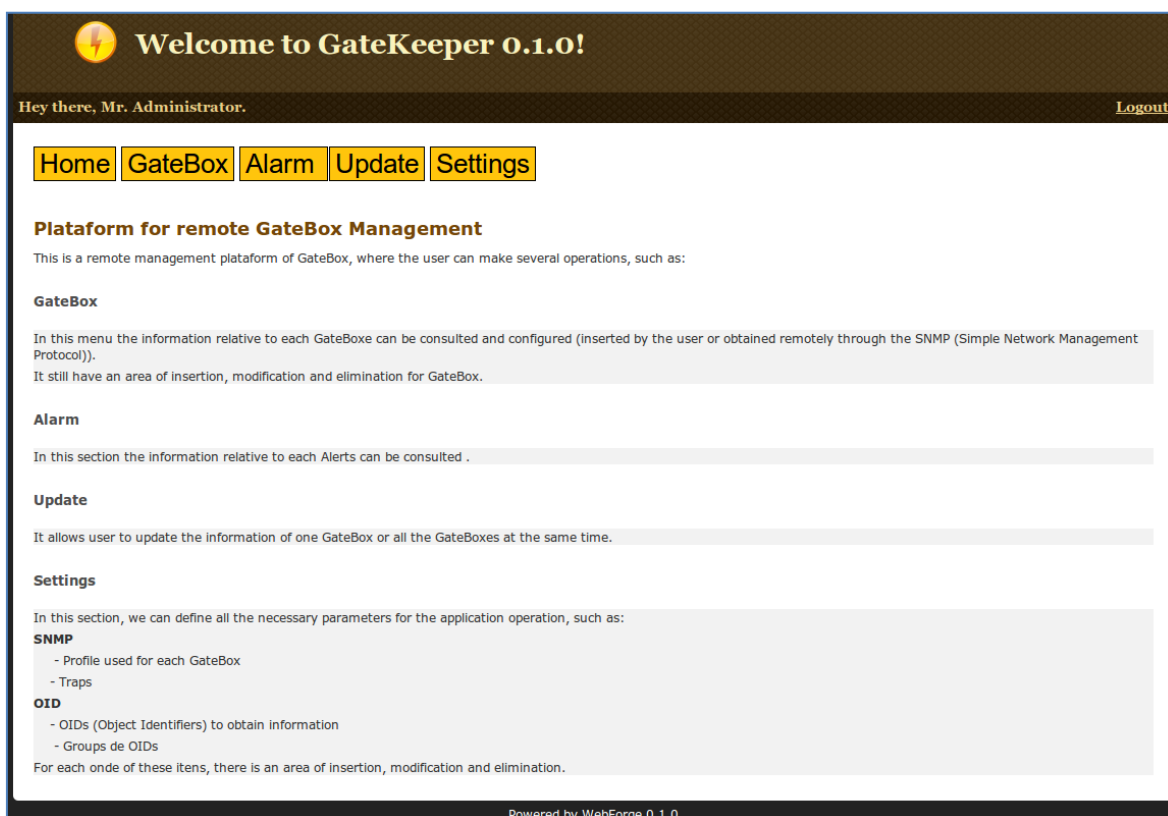



Figura 21 – Menu Home.

4.3.2. MENU GATEBOX

No menu GateBox é onde o utilizador pode efetuar várias operações sobre as GateBoxes tais como, consultar, inserir, modificar, e eliminar informação a elas associada (Figura 22).

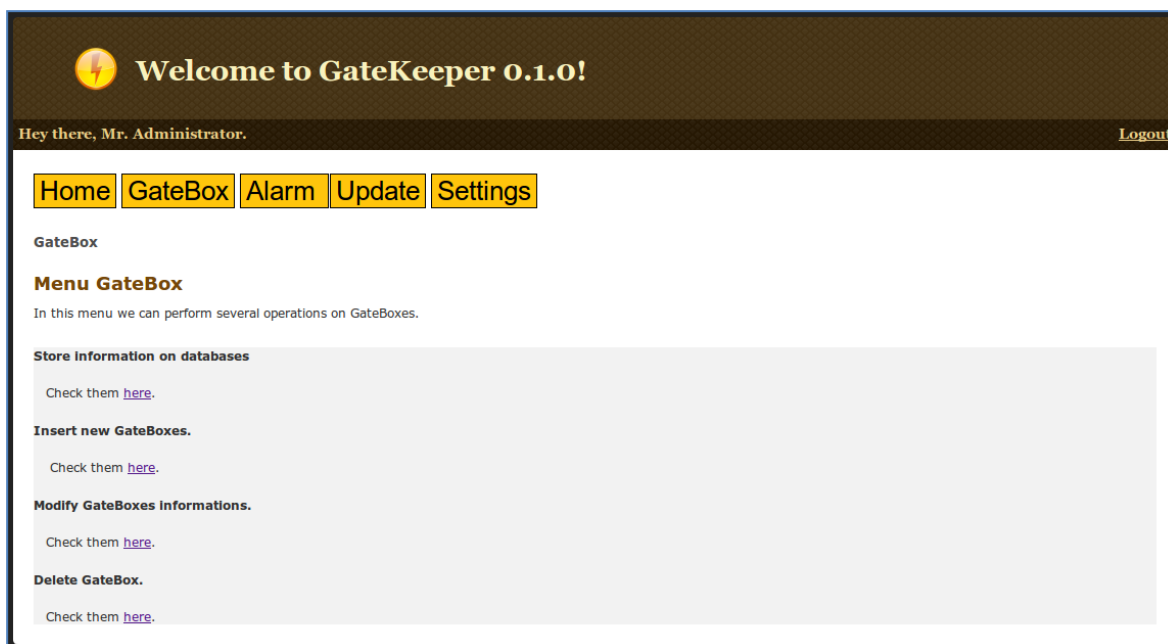



Figura 22 – Menu GateBox.

Na consulta de informação é apresentada uma tabela com todas GateBoxes que se encontram armazenadas na base de dados (Figura 23) com informações inseridas pelo utilizador aquando da sua inserção no sistema.

Gbx ID	Hostname	IP/Domain	Location
1	ubuntu	127.0.0.1	pc_vitor
2	vm_ubuntu	192.168.147.128	Rua de S.Tome 103 3d, 4200-489 Porto
3	gbx_3	10.0.32.210	Instituto Superior de Engenharia do Porto

Figura 23 – Consulta no menu GateBox.

Para consultar mais informações sobre as GateBoxes de uma forma individual deverá aceder-se a uma nova área que permite visualizar, por exemplo, os dados inseridos pelo utilizador, o estado atual da ligação (com um sistema de cores, vermelho para *down* e verde para *up*), e as informações obtidas através do protocolo SNMP. Para isso é necessário clicar sobre o campo *identifier* (ID) correspondente à GateBox que pretende visualizar. Nessa nova página as informações obtidas por SNMP encontram-se separadas por grupos de modo a facilitar a interpretação do seu conteúdo, como se ilustra na Figura

24. Ainda nesta secção visualizar a categoria a que pertence a GateBox, a MIB que utiliza, a versão SNMP (perfil) e data em que esta entrou em funcionamento.

Figura 24 – Consulta detalhada no menu GateBox.

No menu inicial de consulta de GateBox o utilizador pode aceder através da morada da GateBox a uma nova página que onde é possível visualizar através da utilização do *Google Maps* a localização da GateBox [38], como ilustra a Figura 25. Para isso foi utilizada a *Application Programming Interface* (API) disponibilizada gratuitamente pela *Google* para a criação de mapas [37].

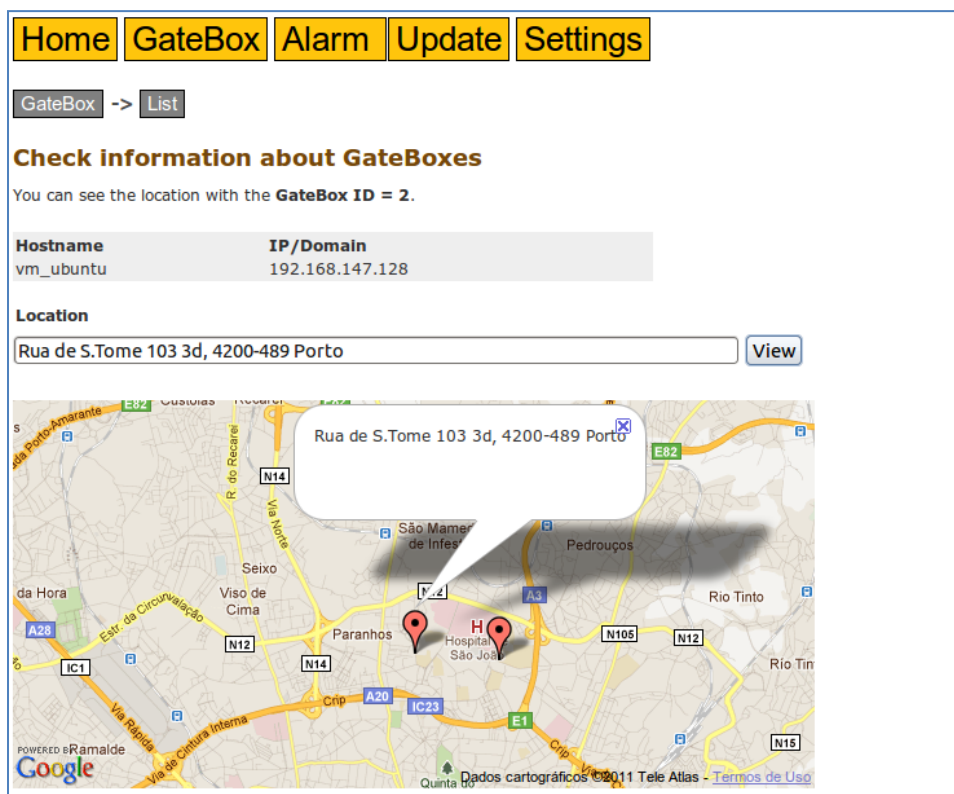



Figura 25 – Consulta da morada no *Google Maps* no menu GateBox.

Na inserção de informação sobre GateBoxes é fornecido ao utilizador um formulário para preenchimento, Figura 26. Nesse formulário devem ser preenchidos os campos, *hostname* para a atribuição de um nome a cada GateBox, o IP/*domain* do dispositivo para a sua identificação e monitorização, a categoria a que pertence, a MIB que utiliza no SNMP, a data do início de funcionamento e por fim, o preenchimento de um texto com a localização do dispositivo. De salientar que todos os dados inseridos serão armazenados nas respetivas tabelas da base de dados.

Figura 26 – Inserção no menu GateBox.

Neste menu pode-se ainda modificar ou eliminar informações sobre as GateBoxes. Ao aceder a estas secções visualizam-se todos os dispositivos armazenados na base de dados, como se ilustra na Figura 27. E assim permitir a escolha da qual se deseja efetuar a modificação, e no caso de se pretender eliminar uma GateBox basta selecionar o dispositivo.

Figura 27 – Modificação no menu GateBox.

Para modificar informações de uma GateBox o utilizador vai, após a escolha, ser redirecionado para um formulário, ilustrado na Figura 28, idêntico ao da secção inserir, só que preenchido com os dados armazenados na base de dados.

Figura 28 – Modificação no menu GateBox (2).

4.3.3. MENU ALARM

No menu Alarm o utilizador pode aceder a todas as notificações provenientes das GateBoxes. Estas são apresentadas numa tabela consoante a ordem de receção (Figura 29). Nesta primeira abordagem apenas constam as informações gerais das notificações, como o seu ID, a GateBox que enviou o alerta, o tipo de notificação e a sua prioridade, que é definida no menu de configurações descrito no ponto 4.3.5.

Home | GateBox | Alarm | Update | Settings

Alarm

GateBox Alerts

In this section you have acess to all database Alarm.
For more detailed information about each Alarm, click on ID.

Alarm ID	Date	GateBox ID	Type	Priority
19	2011-10-06 00:01:13	1	Link Down	Higher
18	2011-08-01 19:35:46	1	Cold Start	Normal
17	2011-08-01 19:35:41	1	Cold Start	Normal
16	2011-08-01 19:35:36	1	Cold Start	Normal
15	2011-08-01 19:35:31	1	Cold Start	Normal
14	2011-08-01 19:35:25	1	Cold Start	Normal
13	2011-08-01 19:35:20	1	Cold Start	Normal
12	2011-08-01 19:35:15	1	Cold Start	Normal
11	2011-08-01 19:35:10	1	Cold Start	Normal
10	2011-08-01 19:35:05	1	Cold Start	Normal
9	2011-08-01 19:35:00	1	Cold Start	Normal
8	2011-08-01 19:34:54	1	Cold Start	Normal
7	2011-08-01 19:34:49	1	Cold Start	Normal
6	2011-08-01 19:34:44	1	Link Down	Higher
5	2011-08-01 19:34:39	1	Cold Start	Normal
4	2011-08-01 19:34:34	1	Cold Start	Normal
3	2011-08-01 19:34:29	1	Link Down	Higher
2	2011-08-01 19:32:52	2	Cold Start	Normal
1	2011-08-01 19:32:44	2	Cold Start	Normal

Figura 29 – Consulta de notificações no menu Alarm.

Ao clicar no ID da notificação o utilizador será redirecionado para uma nova área onde poderá se visualizar mais informações, como se pode constatar na Figura 30. Nesta pode-se visualizar um resumo da informação da notificação, no caso da figura, a interface que foi ativada (`interface vmnet8 up`) e a informação recebida e armazenada no ficheiro pela notificação (`Trap Notification`). O formato de organização da informação foi definido por configuração do *daemon snmptrapd*.

Figura 30 – Consulta detalhada de uma notificação no menu Alarm.

4.3.4. MENU UPDATE

O sistema GateKeeper desenvolvido possui um *script* de obtenção de informações das GateBoxes através do protocolo SNMP. Estas informações são obtidas de uma forma periódica (usando o *cron*) e sem que haja a necessidade da interface Web estar ativa. Contudo o utilizador pode efetuar a recolha de informação também em tempo real através da interface. O menu Update permite ao utilizador duas operações: obter informações de uma GateBox ou então de todas elas, como é ilustrado Figura 31.

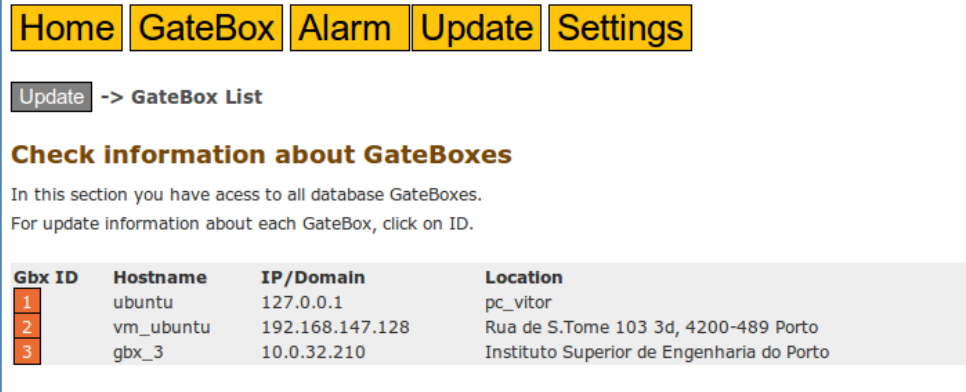



Figura 31 – Menu Update.

No caso de o utilizador optar por atualizar/obter informações de um dispositivo, será redirecionado para uma nova página que lhe proporcionará a escolha da GateBox através do seu ID. O formato de apresentação é semelhante ao da consulta no menu GateBox, Figura 32. No caso de o utilizador optar por atualizar/obter de todos dispositivos, basta só clicar nessa secção.

Figura 32 – Escolha de uma GateBox no menu Update.

4.3.5. MENU SETTINGS

O menu Settings permite ao utilizador efetuar todas as configurações necessárias para o sistema. Este encontra-se dividido em dois submenus principais: o SNMP e o OID, que por sua vez, também estão divididos. No submenu SNMP, a subsecção Profile possibilita a configuração do perfil utilizado no protocolo SNMP, e a definição da *password*, no caso de se utilizar autenticação nos pedidos efetuados. Pode-se ainda nesta secção efetuar a consulta de perfis existentes, inserção, modificação e eliminação de perfis, como se pode verificar na Figura 33.

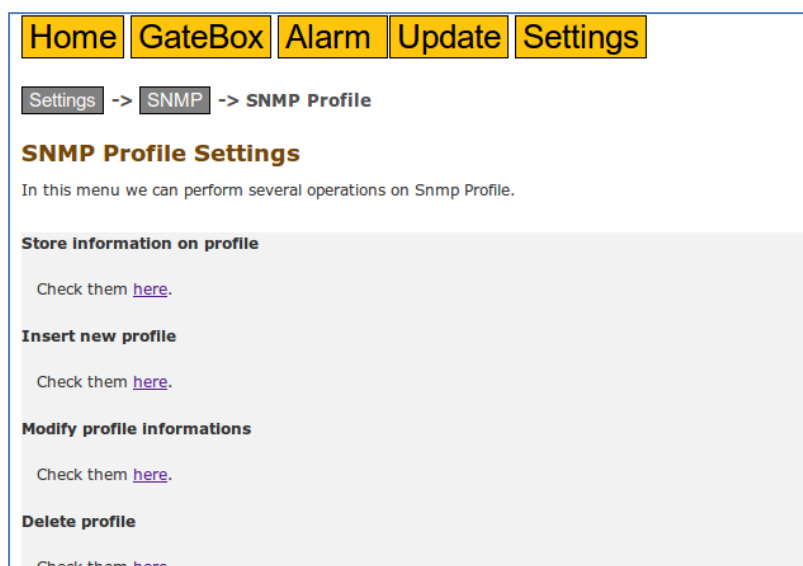



Figura 33 – SNMP Profile do menu Settings.

A interface de interação com o utilizador para estas operações é idêntica à do menu GateBox (cada operação tem uma secção própria, onde na consulta se visualizam todos os perfis, na inserção preenche-se um formulário, etc.), como se pode constatar na Figura 34.

Figura 34 – Consulta de perfis no SNMP Profile no menu Settings.

Na subsecção Trap o utilizador pode configurar os tipos de notificação que deseja analisar e armazenar, e os *emails* para envio dos alertas no caso de novas notificações. Esta subsecção é dividida em duas áreas distintas: Type e Email, para que em cada uma delas se possam realizar diversas operações (consulta, inserção, modificação e eliminação). Na divisão Type, bem como na subsecção Profile, o mecanismo de interação do utilizador na interface é o mesmo para essas operações. Na Figura 35 pode-se visualizar um campo para o nome, um campo de seleção YES ou NO, que permite ao utilizador filtrar as notificações consoante o seu tipo, sendo estas analisadas e armazenadas. Além disso, também se pode definir e escolher a sua prioridade.

Figura 35 – Inserir *trap* em SNMP Trap no menu Settings.

Após essa análise e armazenamento será enviada uma notificação por correio eletrónico para os *emails* que estejam na base de dados. A configuração do *email* é feita na subdivisão Email, onde o utilizador pode consultar, inserir, modificar e eliminar *emails*. Ao contrário dos outros menus, submenus, divisões, estas operações encontram-se todas apresentadas na mesma secção, ou seja na mesma página, Figura 36. Essa diferenciação foi efetuada, devido ao facto de neste caso só estarmos a trabalhar com um dado, o *email*.

Figura 36 – Inserção de emails em SNMP Trap no menu Settings.

O submenu OID permite ao utilizador definir informações sobre os OIDs a monitorizar. Este encontra-se organizado em duas divisões: OID e Group, Figura 37. Em cada uma dessas divisões pode-se realizar várias operações (consultar, inserir, modificar e eliminar).

O processo de interação do utilizador com estas operações é idêntico aos menus e submenus descritos anteriormente.

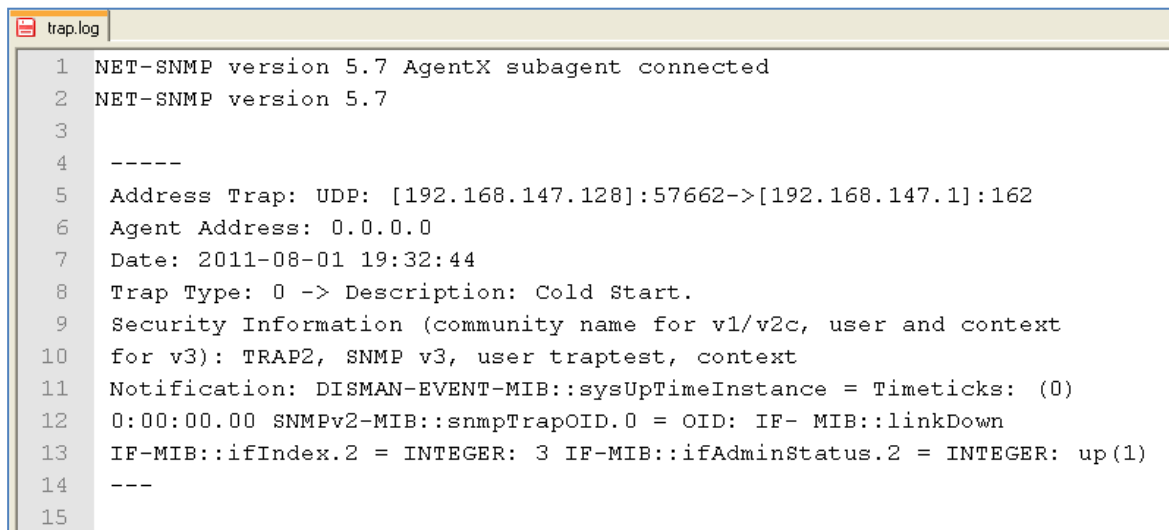
Figura 37 – Submenu OID do menu Settings.

Na Figura 38 podem visualizar-se os dados necessários para definir um OID, tais como: a sequência de números que definem o OID utilizado no pedido SNMP, o nome e a descrição para que utilizador tenha conhecimento da sua funcionalidade quando consultado, o campo *display* utilizado para visualização e identificação do OID na interface Web. O utilizador deve ainda definir o grupo, o método e a MIB a utilizar para este OID.

Figura 38 – Inserção de OID no menu Setting.

4.4. *DAEMONS*

O *daemon* é um tipo de programa usual em sistemas operativos Unix, normalmente executado em segundo plano. Estes programas estão à escuta da ocorrência de um evento ou de alguma condição específica, para se ativarem e desempenhar as funções que lhe foram atribuídas [33]. Os *daemons* configurados para esta aplicação desenvolvida foram três: o `snmpd`, o `snmptrapd` e `crontab`. Os dois primeiros *daemons* são utilizados no protocolo SNMP na realização dos pedidos aos dispositivos e recebimento de notificações. Enquanto o último realiza funções de periodicidade dos *scripts*.

O *daemon* `snmpd` é um agente SNMP que se encontra nos dispositivos a serem monitorizados (GateBoxes), estando à escuta de pedidos por parte da estação de monitorização. Ao receber uma solicitação ele vai processar o pedido e consoante a análise deste, executa a operação solicitada e envia uma resposta ao remetente. No ficheiro de configuração correspondente ao *daemon* `snmpd` foram criados utilizadores com diferentes características de acesso e permissões (sem *password*, com *password*, com *password* e encriptação) para utilização do SNMPv3. E ainda foi configurada a informação do sistema, o local e contacto do dispositivo (`syslocation` e `syscontact`). Na Figura 39 visualizam-se excertos do ficheiro `snmpd.conf`, com as configurações efetuadas.

```
23 #  SNMPv3 AUTHENTICATION
24 #
25 #  Note that these particular settings don't actually belong here.
26 #  They should be copied to the file /var/lib/snmp/snmpd.conf
27 #     and the passwords changed, before being uncommented in that file *only*.
28 #  Then restart the agent
29
30 createUser user1
31 createUser user2 MD5 user2password
32 createUser user3 MD5 user3password DES user3encryption
```

```
65 #rwuser   authPrivUser   priv
66
67 rouser user1 noauth 1.3.6.1.2.1.1
68 rouser user2 auth 1.3.6.1.2.1
69 rwuser user3 priv 1.3.6.1.2.1
70
```

```
78 #  SYSTEM INFORMATION
79 #
80
81 #  Note that setting these values here, results in the corresponding MIB objects being 'read-only'
82 #  See snmpd.conf(5) for more details
83
84 syslocation PC_Home
85 syscontact Vitor
86                                        # Application + End-to-End layers
```

Figura 39 - Ficheiro snmpd.conf

Para que o sistema desenvolvido receba notificações por parte dos dispositivos monitorizados, foi necessária a configuração do *daemon* `snmptrapd` no servidor da estação de gestão. No caso da aplicação desenvolvida, este *daemon* além de receber as notificações, também efetua o registo dos mesmos num ficheiro. Para que a apresentação da notificação no ficheiro seja legível e organizado para o utilizador, foi necessário configurar o *daemon*.

Na Figura 40 pode-se visualizar a informação de uma notificação registada no ficheiro `trap.log`. Nessa notificação pode-se visualizar o endereço de origem da notificação, o endereço da receção, os portos e o protocolo utilizados, a data da ocorrência do evento, o tipo de notificação (ID e descrição), a informação da segurança (versão SNMP, comunidade) utilizada no evento, e por fim, a informação geral da notificação. É neste último parâmetro que o utilizador, após a análise dos dados, pode verificar qual o real problema associado a esse evento. A análise dos dados das notificações é efetuada através de um *script* (analisado no ponto 4.5). No caso da notificação da Figura 40 e após a análise (*script*), o utilizador sabe que ocorreu um *link down* na interface com o índice 4 (`vmnet1`).

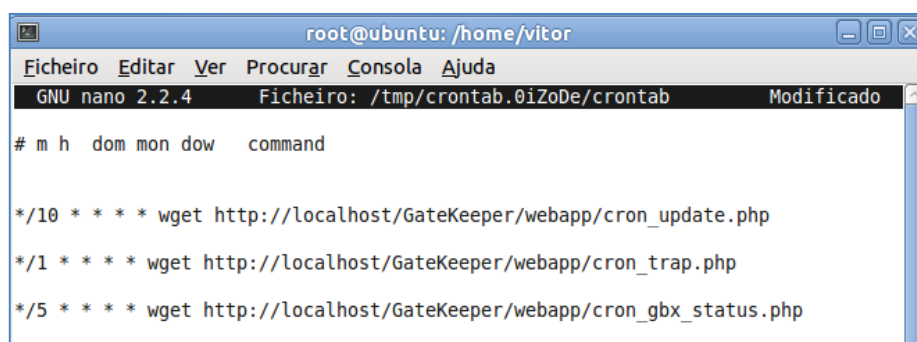
trap.log

```
 1  NET-SNMP version 5.7 AgentX subagent connected
 2  NET-SNMP version 5.7
 3
 4   -----
 5   Address Trap: UDP: [192.168.147.128]:57662->[192.168.147.1]:162
 6   Agent Address: 0.0.0.0
 7   Date: 2011-08-01 19:32:44
 8   Trap Type: 0 -> Description: Cold Start.
 9   Security Information (community name for v1/v2c, user and context
10   for v3): TRAP2, SNMP v3, user traptest, context
11   Notification: DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (0)
12   0:00:00.00 SNMPv2-MIB::snmpTrapOID.0 = OID: IF- MIB::linkDown
13   IF-MIB::ifIndex.2 = INTEGER: 3 IF-MIB::ifAdminStatus.2 = INTEGER: up(1)
14   ---
15
```

Figura 40 – Ficheiro `trap.log`.

A periodicidade dos *scripts* desenvolvidos para aplicação é efetuada pelo "*Cron Jobs*", programa do sistema operativo Linux, que realiza tarefas especificadas pelo utilizador, em determinados momentos ou intervalos de tempo configurados no `crontab` (ficheiro de texto onde é armazenada a informação do *cron* de cada utilizador) [8]. Para configurar o

ficheiro `crontab` inicialmente deve ser a aberto o ficheiro com o comando na consola do sistema operativo: `crontab -e`.

Cada linha do `crontab` tem o seguinte formato:

[minutos] [horas] [dias do mês] [mês] [dias da semana] [comando]

Todos os campos são definidos com valores numéricos, exceção do comando. Na Tabela 1 visualizam-se os valores que podem ser inseridos nestes campos.

Tabela 1 – Valores do crontab.

Campo	Valor
Minutos	0-59
Horas	1-24
Dias do mês	1-31
Mês	1-12
Dias da semana	0-6 (“0” para domingo e “6” para sábado)

Para configuração deste *daemon*, foi editado o ficheiro `crontab` onde se configuram os comandos a serem executados, a hora e o dia da execução do *cron*. Para os três *scripts* desenvolvidos, foi definido no arquivo `crontab` uma periodicidade, como se constata na Figura 41, onde se pode verificar por exemplo, que o *script* `cron_update.php` é executado de 10 em 10 minutos (`*/10 * * * *`).

Figura 41 – Ficheiro `crontab`.

4.5. *SCRIPTS*

Para que o sistema desenvolvido não requeira que a aplicação Web esteja ativa para funcionar, foram criados três *scripts*, um para a atualização periódica das informações das GateBoxes, outro para receção e envio de alerta para o utilizador de notificações (*traps*) e um outro para conferir o estado de cada GateBox.

O *script* de atualização da informação das GateBoxes é utilizado para realizar os pedidos através do protocolo SNMP, entre o servidor e os vários dispositivos. Este *script* denomina-se de `cron_update.php`. O princípio de funcionamento deste é realizar a consulta das GateBoxes que existem na base de dados, e para cada uma das GateBoxes é obtida a informação necessária para a execução do *script*. Inicialmente é efetuado um teste de ligação entre o servidor e o dispositivo monitorizado por SNMP, e para o caso de esta se encontrar inativa, não serão efetuados os comandos SNMP, pois levaria à lentidão do sistema devido aos seus *timeouts*. Após esta verificação e caso a ligação esteja ativa, através da categoria da GateBox, é conhecida a MIB utilizada bem como os seus OIDs. E assim, para cada OID vai efetuar-se o comando e armazenamento na base de dados dos pedidos SNMP ao agente do dispositivo a monitorizar. As informações para a criação do comando (método e perfil SNMP) são obtidas após a consulta à base de dados onde se encontram armazenadas. Os dados resultantes do pedido são analisados e armazenados na base de dados (`data`), separados em campos (`oid`, `type`, `value`). Como este *script* é executado periodicamente, os dados resultantes de um determinado OID já podem constar na base de dados, e nesse caso só é atualizado o valor do `type` e `value` do OID em questão, se não, será criada uma nova linha na tabela `data` para esse OID. No fluxograma da Figura 42 pode-se visualizar de forma esquemática o funcionamento deste *script*.

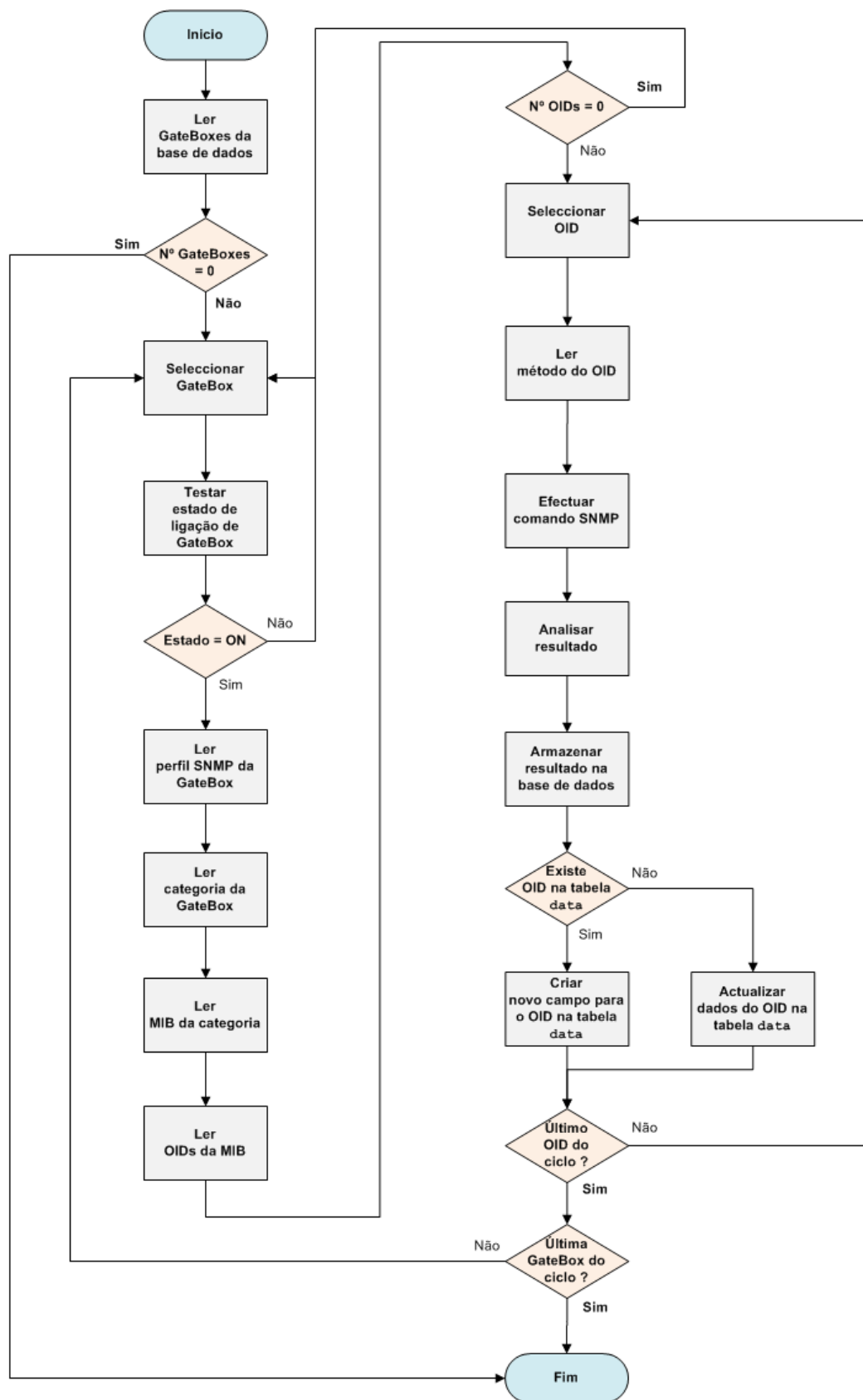



Figura 42 – ***Script*** **`cron_update.php`.**

O *script* cron_trap.php é o responsável pela análise e armazenamento de notificações que o agente do dispositivo monitorizado envia para o servidor. Todas as notificações recebidas dos agentes são guardadas num ficheiro. O *script* vai realizar de uma forma periódica leituras a esse ficheiro, para verificação da existência de novas notificações. No caso de essas existirem, será efetuado para cada uma das notificações a análise da informação nela contida. Em seguida, essa informação é armazenada na base de dados para que possa ser visualizada na interface Web, e procede-se também ao envio por correio eletrónico de uma mensagem de alerta para o utilizador/gestor. Na Figura 43 ilustra-se o fluxograma deste *script*.

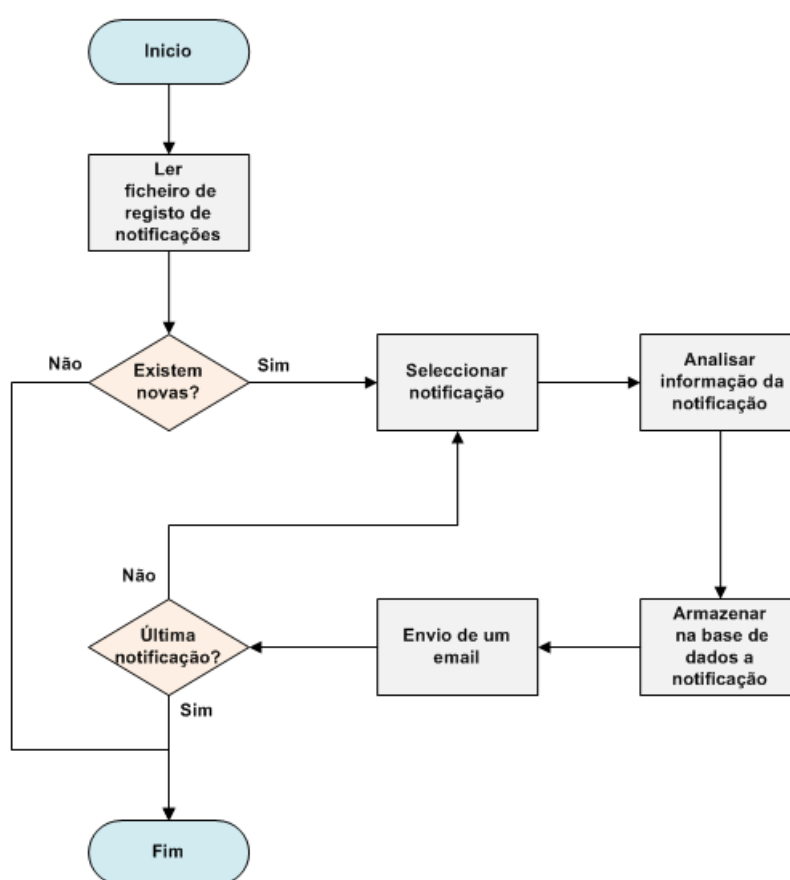



Figura 43 – S*cript* `cron_trap.php`.

O *script* cron_status.php realiza um teste à ligação entre o sistema GateKeeper e os dispositivos a monitorizar. Inicialmente vai-se realizar uma consulta das GateBoxes existentes, obtendo-se o endereço IP de cada uma delas, necessário para a efetuar um PING do servidor com o dispositivo. Posteriormente, será efetuada a análise da resposta do

comando executado bem como o armazenamento na base de dados do estado da ligação (*Up*/*Down*). Pode-se visualizar na Figura 44 o fluxograma deste *script*.

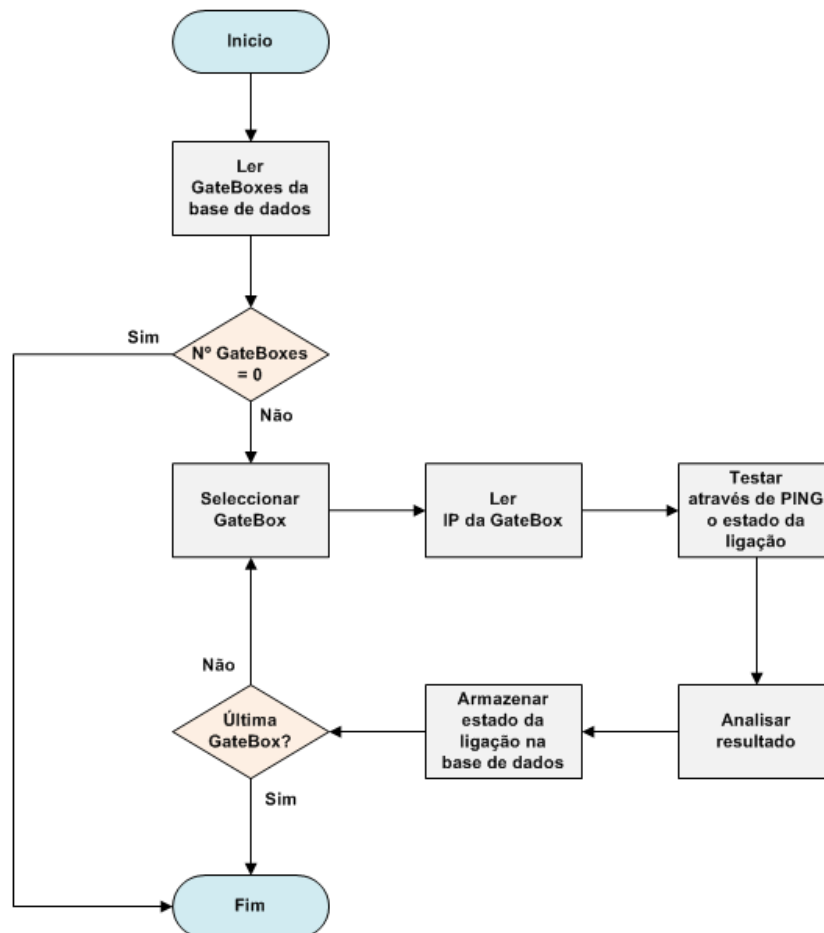



Figura 44 - *Script* `cron_status.php`.

5. TESTES E DEMONSTRAÇÕES

Neste capítulo, serão abordados alguns testes e demonstrações que comprovam o funcionamento da aplicação desenvolvida. Esses testes e demonstrações foram obtidos através de um ambiente emulado, uma vez que a empresa em causa, a NextToYou, não pôde disponibilizar nenhuma GateBox. Sendo assim, foram efetuados apenas testes e demonstrações da aplicação GateKeeper localmente no computador de trabalho, através de VMware para a criação de máquinas virtuais, e também um computador externo para a emulação de GateBoxes.

5.1. TESTE DA APLICAÇÃO WEB

A metodologia e as ferramentas utilizadas para realização do teste da aplicação Web depende das características da aplicação e dos parâmetros de desenvolvimento, tais como as linguagens e *software*. A utilidade que se pretende para a aplicação pode ter um fator de influência. Por exemplo, no caso de serem aplicações bancárias, a segurança assume um papel prioritário maior do que a sua utilidade, ao contrário de uma aplicação na área da saúde em que a disponibilidade e a utilidade são fatores chave [34].

A aplicação Web engloba todos os aplicativos que podem ser acedidos através de um navegador [35]. Ao longo desta secção serão descritas algumas das metodologias de teste utilizadas para avaliar a aplicação Web. Para os vários pontos abordados não se realizaram todos os testes possíveis, visto que o teste exaustivo seria fastidioso de descrever. Assim, optou-se por representar nesta tese apenas um exemplo de cada tipo de teste realizado.

5.1.1. TESTE DE UTILIDADE

Os testes de utilidade de uma aplicação Web avaliam a interação do utilizador com a interface. Uma aplicação Web para ser eficaz para o utilizador deve permitir uma fácil e adequada navegabilidade entre páginas Web. As instruções da interface devem então assim ser transmitidas de uma forma clara. O menu principal deve ser apresentado em todas as páginas da aplicação para uma maior consistência da aplicação. O conteúdo fornecido deve ser lógico e fácil de entender, e deve ser também evitado o excesso de conteúdo numa determinada página. As cores utilizadas devem sempre seguir um padrão. Estas são algumas normas básicas que devem ser utilizadas quando se cria uma aplicação Web [36].

A interface Web do sistema GateKeeper proporciona ao utilizador a interação e realização de determinadas funcionalidades (consulta, inserção, modificação e eliminação de informação). Estas vão ser demonstradas de seguida e de forma individual, podendo-se constatar que esta cumpre as normas básicas de criação de uma aplicação Web.

a) Consulta de informação

A Figura 45 ilustra como se efetua uma consulta das informações de uma GateBox, armazenada na base de dados e como essas são apresentadas na interface Web. Quando o utilizador interage com a secção de consulta do menu GateBox, este vai visualizar inicialmente uma tabela com todas as GateBoxes existentes e os dados gerais destas (inseridos pelo utilizador aquando o registo). Após a escolha de uma delas (através do ID), o utilizador será redirecionado para uma nova página onde poderá visualizar todas as informações armazenadas na base de dados da GateBox escolhida (1). Nesta nova página pode-se visualizar todas as informações, inseridas pelo utilizador e obtidas através do protocolo SNMP. Os dados recolhidos por SNMP estão divididas em grupos, para proporcionarem uma melhor organização dessa informação (2).

Figura 45 – Demonstração do processo de uma consulta na interface Web.

Nesta demonstração pode-se verificar que a aplicação Web cumpre as normas básicas, como por exemplo, a visualização do menu principal em todas as páginas visualizadas, a informação é apresentada de um forma clara e sucinta.

b) INSERÇÃO DE INFORMAÇÃO

Na aplicação Web existem vários formulários para inserção de informações na base de dados. Seguidamente, será testado um registo de uma GateBox, como se ilustra na Figura 46. Na inserção de uma GateBox o utilizador tem que preencher vários campos com dados sobre o equipamento. Depois disso (1), no caso de existirem campos de preenchimento obrigatório sem informação, a inserção não se realizará na base de dados e será apresentada uma mensagem de notificação. Caso contrário, será apresentada uma mensagem de confirmação de registo bem como a informação inserida.

Figura 46 - Demonstração do processo de uma inserção na interface Web

c) Modificação de informação

Na Figura 47 pode-se visualizar uma demonstração do processo de modificação de dados de uma GateBox. O utilizador pode modificar informações na interface que tenha inserido sobre as GateBoxes, OIDs, grupos de OIDs, perfis SNMP, tipos de notificações e *emails* para envio de alertas. Para cada um destes, são apresentados ao utilizador todos os elementos que se encontram na base de dados, permitindo a sua escolha para que seja feita a modificação. Após essa escolha (1), o utilizador será redirecionado para uma nova página com um formulário preenchido com a informação armazenada. É neste formulário que vai ser realizada a alteração dos dados (2). Consequentemente será armazenado na base de dados os dados modificados e será apresentada ao utilizador uma mensagem de confirmação da alteração bem como a informação modificada (3).

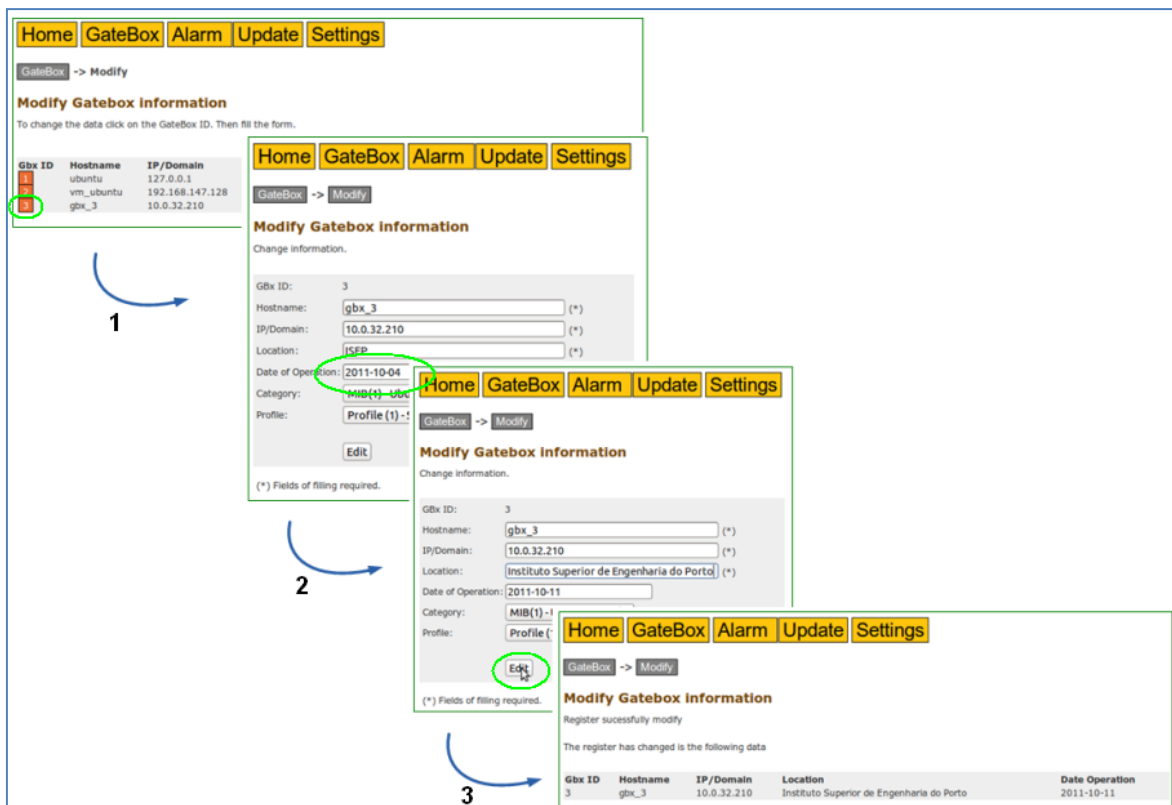



Figura 47 – Demonstração do processo de uma modificação na interface Web.

d) ELIMINAÇÃO DE INFORMAÇÃO

O utilizador caso pretenda pode também eliminar informações na interface Web, informações de GateBoxes, OIDs, entre outros. Na Figura 48 ilustra-se o processo de eliminação de uma GateBox. A apresentação dessas informações é idêntica à da apresentada na modificação, isto é, inicialmente o utilizador tem a opção de escolher qual o elemento que deseja eliminar, e após essa escolha (1), será apresentado ao utilizador uma mensagem de confirmação ou da ocorrência de algum erro na remoção. Em seguida (2), o utilizador é redirecionado para a área inicial da secção de eliminação do menu GateBox.

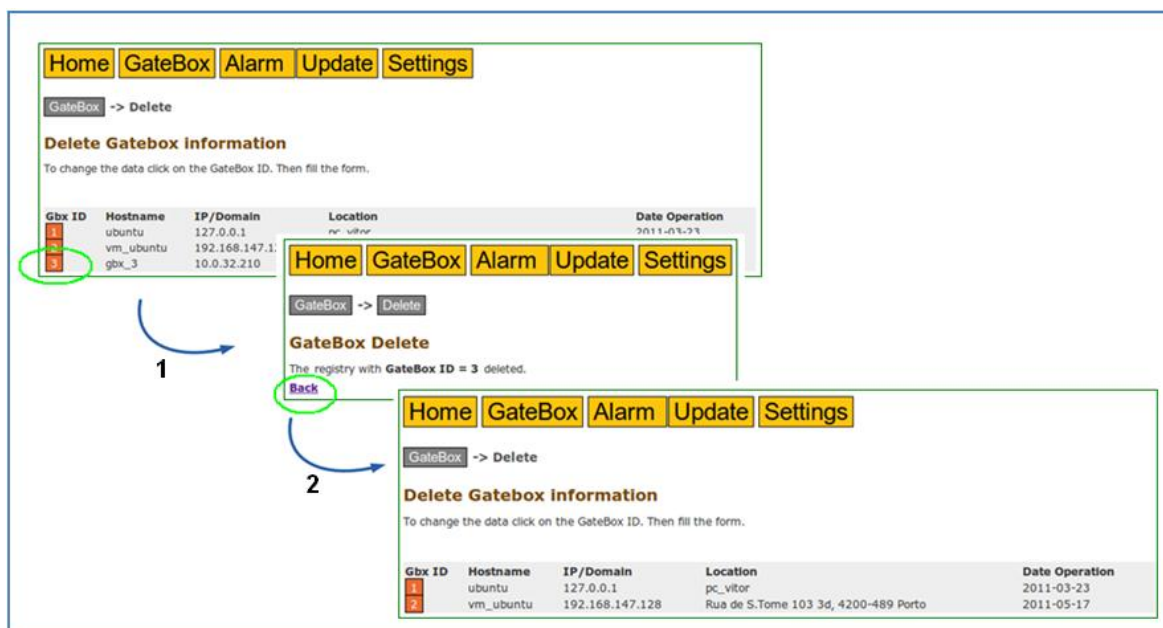



Figura 48 – Demonstração do processo de uma eliminação na interface Web.

5.1.2. TESTE DE FUNCIONALIDADE

O teste de funcionalidade garante que as funções individuais da aplicação Web estejam corretamente implementadas. No caso de erro devem originar mensagens de erro na aplicação ou em ficheiros de registo (*logs*). Os testes de funcionalidade avaliam todos os *links* da aplicação Web, a comunicação com a base de dados, os vários formulários que possam existir na aplicação, os *cookies* e a validação do HTML/CSS da aplicação [36].

No ponto 5.1.1 pode-se visualizar alguns dos vários formulários existentes na aplicação e de que forma estes efetuam a interação do utilizador. Ainda nesse ponto, foram analisadas as funcionalidades existentes no menu GateBox da aplicação Web como: consulta, inserção, modificação e eliminação de informação. A comunicação com a base de dados, dessas funcionalidades, pode ser testada e demonstrada através do ficheiro `Gatekeeper.log` que regista as várias *queries* efetuadas com a base de dados, durante a navegação do utilizador na interface. Nas alíneas seguintes vão ser analisados alguns excertos do ficheiro que demonstram as funcionalidades realizadas na interface em particular e a título de exemplo, no menu GateBox.

a) CONSULTA DE INFORMAÇÃO

Na Figura 49 ilustra-se um excerto do ficheiro de registo (*logs*) da aplicação desenvolvida, onde se constata o princípio de funcionamento de uma consulta à base de dados.

Inicialmente é inicializada a plataforma WebCore e o *template* (1), seguidamente criam-se as ligações com as bases de dados necessárias para a página Web (`created handler`) (2) e a confirmação da sessão do utilizador (`Service running`) (3). Após estes passos podem-se visualizar na Figura 49 todas as *queries* efetuadas corretamente à base de dados de obtenção de informações (`SELECT`) para preenchimento de campos na interface (4).

```
2011-10-05 18:44:36.0777 INFO  (6): ========== New Request : /GateKeeper/webapp/gbx?page=list&gbx_id=1&group=1 ==========
2011-10-05 18:44:36.0778 DEBUG (7): WebCore 0.2.0 started
2011-10-05 18:44:36.0825 DEBUG (7): Template: service running...
2011-10-05 18:44:36.0924 DEBUG (7): DB: created handler 'webcore'
2011-10-05 18:44:36.0928 DEBUG (7): DB: created handler 'gatekeeper'
2011-10-05 18:44:36.0945 DEBUG (7): DB: Query( 1): SELECT DISTINCT * FROM `user` WHERE `user_id`='1'
2011-10-05 18:44:36.0961 DEBUG (7): Icon: service running...
2011-10-05 18:44:36.0972 DEBUG (7): DB: Query( 2): SELECT * FROM `gbx` WHERE gbx_id='1'
2011-10-05 18:44:36.0977 DEBUG (7): DB: Query( 3): SELECT * FROM `gbx` WHERE gbx_id='1'
2011-10-05 18:44:36.0987 DEBUG (7): DB: Query( 4): SELECT * FROM `gbx_model` WHERE model_id='1'
2011-10-05 18:44:36.0992 DEBUG (7): DB: Query( 5): SELECT * FROM `gbx_model` WHERE model_id='1'
2011-10-05 18:44:36.0998 DEBUG (7): DB: Query( 6): SELECT * FROM `mib` WHERE mib_id='1'
2011-10-05 18:44:36.1005 DEBUG (7): DB: Query( 7): SELECT * FROM `snmp_gbx` WHERE gbx_id='1'
2011-10-05 18:44:36.1014 DEBUG (7): DB: Query( 8): SELECT * FROM `profile` WHERE profile_id='4'
2011-10-05 18:44:36.1024 DEBUG (7): DB: Query( 9): SELECT * FROM `group_oid`
2011-10-05 18:44:36.1037 DEBUG (7): DB: Query(10): SELECT * FROM `gbx_status` WHERE gbx_id='1'
2011-10-05 18:44:36.1044 DEBUG (7): Template: started display
2011-10-05 18:44:36.1612 DEBUG (7): Template: started display
2011-10-05 18:44:36.1671 DEBUG (7): DB: Query(11): SELECT * FROM `group_oid` WHERE group_id='1'
2011-10-05 18:44:36.1689 DEBUG (7): DB: Query(12): SELECT * FROM `oid` WHERE group_id='1'
2011-10-05 18:44:36.1697 DEBUG (7): Template: started display
2011-10-05 18:44:36.2295 DEBUG (7): Template: started display
2011-10-05 18:44:36.2473 DEBUG (7): DB: Query(13): SELECT * FROM `oid` WHERE oid_id='1'
2011-10-05 18:44:36.2484 DEBUG (7): DB: Query(14): SELECT * FROM `property` WHERE property_id='1'
2011-10-05 18:44:36.2496 DEBUG (7): DB: Query(15): SELECT * FROM `data` WHERE gbx_id='1' AND oid_id='1'
2011-10-05 18:44:36.2503 DEBUG (7): Template: started display
2011-10-05 18:44:36.2683 DEBUG (7): DB: Query(16): SELECT * FROM `oid` WHERE oid_id='2'
2011-10-05 18:44:36.2688 DEBUG (7): DB: Query(17): SELECT * FROM `property` WHERE property_id='2'
```

Figura 49 – *Log* de consulta da GateBox.

b) Inserção de informação

Na Figura 50 ilustram-se todas as *queries* efetuadas à base de dados na inserção de informações, no caso desta figura, de uma GateBox. Temos as *queries* `INSERT INTO` para inserir informação nas tabelas da base de dados (1 e 3), e as `SELECT` para seleção do último registo efetuado, para confirmação do registo e visualização do mesmo na interface (2).

```
2011-10-05 22:42:45.0815 INFO  (6): ========== New Request : /GateKeeper/webapp/gbx?page=insert ==========
2011-10-05 22:42:45.0817 DEBUG (7): WebCore 0.2.0 started
2011-10-05 22:42:45.0870 DEBUG (7): Template: service running...
2011-10-05 22:42:45.1021 DEBUG (7): DB: created handler 'webcore'
2011-10-05 22:42:45.1027 DEBUG (7): DB: created handler 'gatekeeper'
2011-10-05 22:42:45.1046 DEBUG (7): DB: Query( 1): SELECT DISTINCT * FROM `user` WHERE `user_id`='1'
2011-10-05 22:42:45.1063 DEBUG (7): Icon: service running...
2011-10-05 22:42:45.1078 DEBUG (7): DB: Query( 2): INSERT INTO `gbx` (hostname,ip_domain,location,date_operation,
  model_id,created, modified, modified_by ) VALUES ('gbx_3','10.0.32.210','ISEP','2011-10-04','1',
  '2011-10-05 22:42:45','2011-10-05 22:42:45','1' )
2011-10-05 22:42:45.1084 DEBUG (7): DB: Query( 3): SELECT LAST_INSERT_ID()
2011-10-05 22:42:45.1088 DEBUG (7): DB: Query( 4): SELECT * FROM `gbx` WHERE gbx_id='3'
2011-10-05 22:42:45.1093 DEBUG (7): DB: Query( 5): SELECT LAST_INSERT_ID()
2011-10-05 22:42:45.1238 DEBUG (7): DB: Query( 6): INSERT INTO `snmp_gbx` (gbx_id,profile_id) VALUES ('3','1')
2011-10-05 22:42:45.1246 DEBUG (7): Profiler: [Runtime: 0.042 sec, 100.0% PHP, 0.0% Template] [DB: 6 queries]
```

Figura 50 – *Log* da inserção de uma GateBox.

c) MODIFICAÇÃO DE INFORMAÇÃO

No ficheiro de *logs* (Figura 51), são registadas todas as *queries* efetuadas numa modificação realizada pelo utilizador. As informações armazenadas sobre o elemento selecionado são apresentadas no formulário para que sejam feitas as alterações. Essas informações são obtidas através de pedidos à base de dados (`SELECT`) (1). Após a modificação, é efetuado o armazenamento na base de dados (`UPDATE`) das informações atuais (2).

```
2011-10-05 22:44:05.6781 INFO (6): ========== New Request : /GateKeeper/webapp/gbx?page=modify&gbx_id=3 ==========
2011-10-05 22:44:05.6783 DEBUG (7): WebCore 0.2.0 started
2011-10-05 22:44:05.6848 DEBUG (7): Template: service running...
2011-10-05 22:44:05.6950 DEBUG (7): DB: created handler 'webcore'
2011-10-05 22:44:05.6957 DEBUG (7): DB: created handler 'gatekeeper'
2011-10-05 22:44:05.6977 DEBUG (7): DB: Query( 1): SELECT DISTINCT * FROM `user` WHERE `user_id`='1'
2011-10-05 22:44:05.6995 DEBUG (7): Icon: service running...
2011-10-05 22:44:05.7008 DEBUG (7): DB: Query( 2): SELECT * FROM `gbx` WHERE gbx_id='3'
2011-10-05 22:44:05.7019 DEBUG (7): DB: Query( 3): SELECT * FROM `gbx_model`
2011-10-05 22:44:05.7033 DEBUG (7): DB: Query( 4): SELECT * FROM `snmp_gbx` WHERE gbx_id='3'
2011-10-05 22:44:05.7048 DEBUG (7): DB: Query( 5): SELECT * FROM `profile`
2011-10-05 22:44:05.7053 DEBUG (7): Template: started display
2011-10-05 22:44:05.8734 DEBUG (7): Profiler: [Runtime: 0.194 sec, 13.6% PHP, 86.4% Template] [DB: 5 queries]
2011-10-05 22:45:11.0401 INFO (6): ========== New Request : /GateKeeper/webapp/gbx?page=modify&gbx_id=3&num=2 ========
2011-10-05 22:45:11.0402 DEBUG (7): WebCore 0.2.0 started
2011-10-05 22:45:11.0449 DEBUG (7): Template: service running...
2011-10-05 22:45:11.0550 DEBUG (7): DB: created handler 'webcore'
2011-10-05 22:45:11.0555 DEBUG (7): DB: created handler 'gatekeeper'
2011-10-05 22:45:11.0572 DEBUG (7): DB: Query( 1): SELECT DISTINCT * FROM `user` WHERE `user_id`='1'
2011-10-05 22:45:11.0588 DEBUG (7): Icon: service running...
2011-10-05 22:45:11.0603 DEBUG (7): DB: Query( 2): UPDATE `gbx` SET hostname='gbx_3', ip_domain='10.0.32.210',
location='Instituto Superior de Engenharia do Porto', date_operation='2011-10-11',model_id='1 = 1',
 modified='2011-10-05 22:45:11', modified_by='1' WHERE gbx_id='3'
2011-10-05 22:45:11.0610 DEBUG (7): DB: Query( 3): SELECT * FROM `gbx` WHERE gbx_id='3'
2011-10-05 22:45:11.0623 DEBUG (7): DB: Query( 4): UPDATE `snmp_gbx` SET profile_id='1' WHERE gbx_id='3'
2011-10-05 22:45:11.0629 DEBUG (7): DB: Query( 5): SELECT * FROM `snmp_gbx` WHERE gbx_id='3'
2011-10-05 22:45:11.0635 DEBUG (7): Template: started display
2011-10-05 22:45:11.1668 DEBUG (7): Profiler: [Runtime: 0.126 sec, 18.2% PHP, 81.8% Template] [DB: 5 queries]
```

Figura 51 – *Log* da modificação de informações de uma GateBox.

d) ELIMINAÇÃO DE INFORMAÇÃO

Na Figura 52 ilustra-se as *queries* realizadas no ficheiro de *logs* na eliminação de um GateBox, são inicialmente idênticas às da modificação de um elemento (`SELECT`) (1). A obtenção de informação armazenada na base de dados originará a página de seleção do elemento a eliminar. Após a escolha desse elemento este será removido da base de dados através da *query* `DELETE` (2).

```
2011-10-05 22:47:42.7068 INFO (6): ========== New Request : /GateKeeper/webapp/gbx?page=delete ==========
2011-10-05 22:47:42.7069 DEBUG (7): WebCore 0.2.0 started
2011-10-05 22:47:42.7114 DEBUG (7): Template: service running...
2011-10-05 22:47:42.7227 DEBUG (7): DB: created handler 'webcore'
2011-10-05 22:47:42.7232 DEBUG (7): DB: created handler 'gatekeeper'
2011-10-05 22:47:42.7253 DEBUG (7): DB: Query( 1): SELECT DISTINCT * FROM `user` WHERE `user_id`='1'
2011-10-05 22:47:42.7272 DEBUG (7): Icon: service running...
2011-10-05 22:47:42.7283 DEBUG (7): DB: Query( 2): SELECT * FROM `gbx`                              1
2011-10-05 22:47:42.7291 DEBUG (7): Template: started display
2011-10-05 22:47:42.8263 DEBUG (7): Profiler: [Runtime: 0.119 sec, 18.4% PHP, 81.6% Template] [DB: 2 queries]
2011-10-05 22:47:54.0846 INFO (6): ========== New Request : /GateKeeper/webapp/gbx?page=delete&gbx_id=3 ==========
2011-10-05 22:47:54.0848 DEBUG (7): WebCore 0.2.0 started
2011-10-05 22:47:54.0893 DEBUG (7): Template: service running...
2011-10-05 22:47:54.1008 DEBUG (7): DB: created handler 'webcore'
2011-10-05 22:47:54.1013 DEBUG (7): DB: created handler 'gatekeeper'
2011-10-05 22:47:54.1032 DEBUG (7): DB: Query( 1): SELECT DISTINCT * FROM `user` WHERE `user_id`='1'
2011-10-05 22:47:54.1049 DEBUG (7): Icon: service running...
2011-10-05 22:47:54.1059 DEBUG (7): DB: Query( 2): DELETE FROM `gbx` WHERE `gbx_id` = '3'             2
2011-10-05 22:47:54.1064 DEBUG (7): Template: started display
2011-10-05 22:47:54.2077 DEBUG (7): Profiler: [Runtime: 0.122 sec, 17.4% PHP, 82.6% Template] [DB: 2 queries]
```

Figura 52 – *Log* da eliminação de uma GateBox.

5.1.3. TESTE DE INTERFACE

Nesta área pode-se efetuar três testes: à aplicação, ao servidor Web e ao servidor de base de dados. Para comprovar o correto funcionamento da aplicação podem-se enviar pedidos para a base de dados e visualizar a saída desses dados na interface Web. O servidor Web e o servidor de base dados podem ser testados verificando se estes executam todas as instruções corretamente. Além disso, pode-se realizar uma análise de como a aplicação Web se comporta quando um destes servidores não tiveram conexão, ou esta reposta estiver a meio de uma tarefa inicializada anteriormente [39].

Os testes de funcionamento da aplicação na interface podem incluir os pontos 5.1.1 e 5.1.2, onde já foi feita uma descrição e análise do seu desempenho. Ainda nesses pontos foram efetuados alguns testes que comprovam a execução correta de instruções, por exemplo as *queries* à base de dados. Para testar os comportamentos dos servidores no caso de não existir conexão, desativou-se individualmente o servidor Web e em seguida o servidor de base de dados. No primeiro caso, a aplicação não é executada, surgindo uma informação no *browser* de como o servidor se encontra sem conexão. No segundo caso, surgirá na aplicação uma mensagem (Figura 53) a informar que a ligação ao servidor de base de dados não é possível.

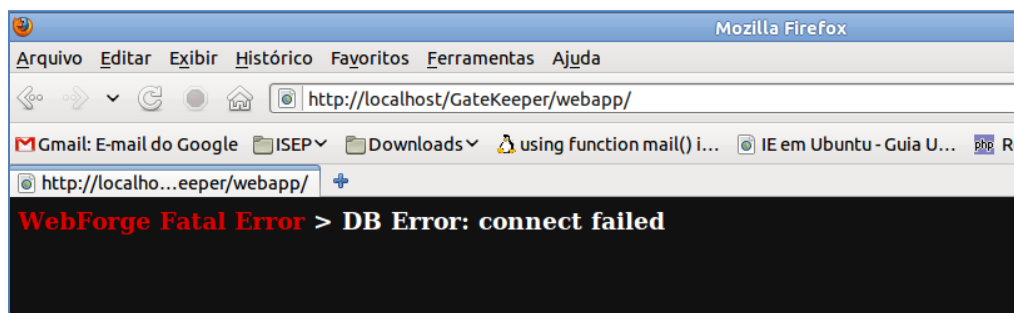



Figura 53 – Mensagem de erro na conexão com o servidor de base de dados.

5.1.4. TESTE DE COMPATIBILIDADE

A compatibilidade de uma aplicação Web é um aspeto importante para o utilizador, visto que pode afetar o bom funcionamento da aplicação. Por exemplo, a falta de compatibilidade com diferentes *browsers* pode originar que certas funcionalidades da aplicação possam não funcionar corretamente, assim como a compatibilidade da aplicação com vários sistemas operativos, a navegação em dispositivos móveis e opções de impressão de páginas da aplicação [36].

A plataforma de desenvolvimento sugerida e fornecida pela NextToYou para a realização do trabalho deve ser executada em ambiente Linux. Assim, o servidor do sistema desenvolvido fica limitado em termos de compatibilidade de sistemas operativos.

Como referido anteriormente, a utilização de diferentes *browsers* é um aspeto que pode afetar o bom funcionamento da aplicação. Portanto, para testar os diferentes tipos de comportamentos, foram utilizados alguns dos *browsers* mais populares como o *Firefox*, o *Chrome*, o *Opera*, para assim se demonstrar a compatibilidade para com a aplicação. Nestes, foram também testados todos os tipos de funcionalidades que se podem realizar na interface, assim como foi também efetuada uma análise dos aspetos visuais, organizacionais e arquitetónicos da aplicação. Dessa forma pode-se constatar, pela Figura 54, que para qualquer um destes *browsers* a aplicação não sofre nenhuma alteração na sua apresentação.

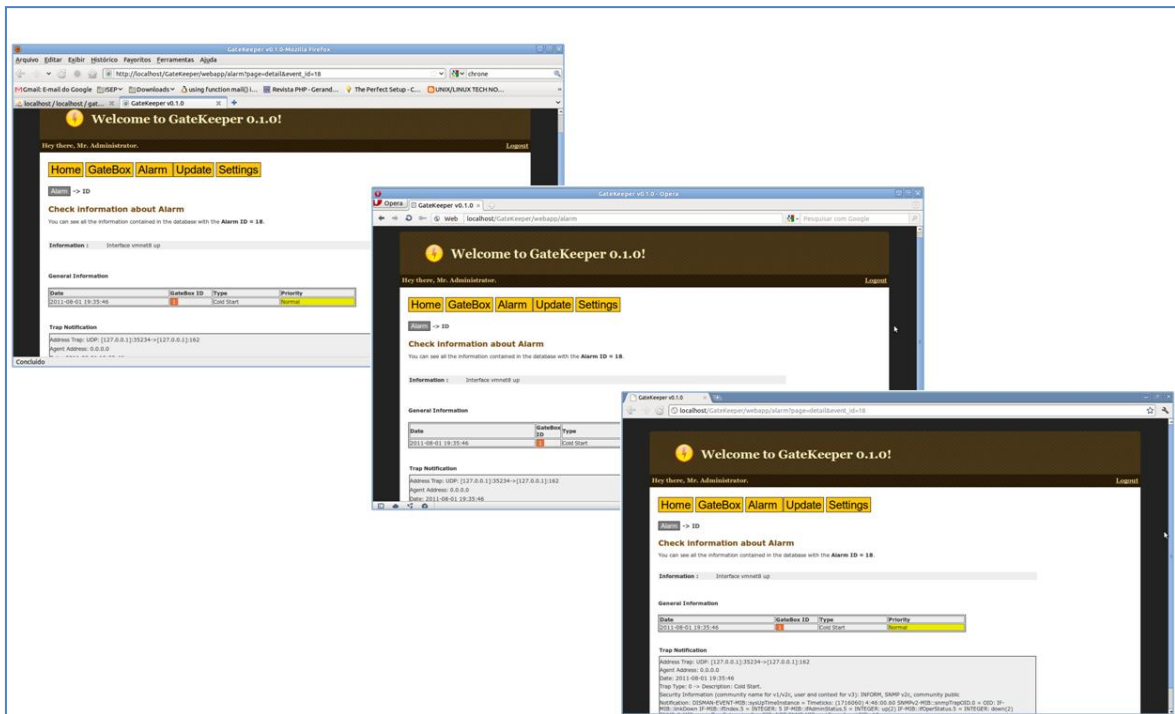



Figura 54 – Demonstração de diversos *browsers*.

5.1.5. TESTE DE DESEMPENHO

Os testes de desempenho em aplicações Web determinam a performance da aplicação em vários cenários. Estes testes devem incluir: testes de *stress* que determinam os limites máximos de resposta da aplicação e testes de carga que proporcionam ao utilizador constatar como o sistema se comporta em ambientes de carga elevada. No teste de carga o utilizador pode obter informações importantes como por exemplo, o rendimento da utilização da *Central Processing Unit* (CPU), a utilização da memória, entre outros [34].

Na aplicação desenvolvida foram realizados testes desempenho da capacidade, verificando o comportamento do servidor (computador com funções da estação de gestão) quando efetuar, após uma ordem inserida no menu Update, uma recolha e armazenamento de informação a todas as GateBoxes. Este menu pode ser equiparado ao *script* de recolha e armazenamento de informações provenientes de pedidos SNMP. Contudo, o teste foi realizado num ambiente emulado, isto é, não foi testado em ambiente real de Internet. Ainda assim, permitiu a obtenção de informações importantes sobre a duração da execução da tarefa. O teste foi realizado num computador com CPU de 1,66 GHz e 2 GB de *Random Access Memory* (RAM), e recolhida a informação de três GateBoxes, o próprio computador, uma máquina virtual e um computador externo. Da realização dessa tarefa foi medido um tempo de 6,672 segundos na execução para os três dispositivos. Contudo, a

execução da mesma tarefa apenas para um dispositivo (o computador externo), com o VMware desligado e com o GateKeeper ativo, obteve-se um tempo de recolha e armazenamento de informação SNMP de 1,437 segundos. Analisando este valor pode-se constatar que houve uma redução do valor médio de execução da tarefa (de ≈ 2,224 segundos para 1,437 segundos de tempo média por dispositivo). Mesmo assim, para um número elevado de unidades de GateBoxes a aplicação poderia ter uma duração elevada na execução da tarefa. Por exemplo, o servidor necessita para cada GateBox aproximadamente de 1,5 segundos para realizar a recolha e armazenamento das informações. Então, para cem GateBoxes demoraria aproximadamente dois minutos e meio, e para mil GateBoxes por volta de vinte e cinco minutos, isto realizando uns cálculos lineares e aproximados aos valores obtidos. Destes valores, pode-se chegar à conclusão que um servidor com as capacidades (CPU e RAM) do utilizado nos testes, a aplicação não será o mais adequado instalar a aplicação GateKeeper na empresa. Além dessas limitações, a utilização de máquinas virtuais ainda eleva a limitação da CPU e da RAM na aplicação. Estima-se que a utilização de um servidor dedicado para execução da aplicação, por parte da empresa, originará uma diminuição dos tempos de execução.

5.1.6. TESTE DE SEGURANÇA

No desenvolvimento de uma aplicação Web um dos fatores mais relevantes é a segurança que proporciona ao utilizador, protegendo a aplicação de ameaças externas. Assim, o teste da segurança de uma aplicação tem uma importância elevada para garantir a fiabilidade da mesma. Para isso, podem ser realizados alguns testes, como por exemplo: inserir um endereço de uma página interna da aplicação na barra de endereços do navegador sem que se tenha efetuado o login, tentar aceder com ID de outro utilizador quando ainda se encontra uma sessão iniciada anteriormente, e verificar o comportamento da aplicação quando se inserem dados inválidos nos campos da aplicação (por exemplo, o login e *password* do utilizador) [34].

Na aplicação desenvolvida foi testada a sessão do utilizador, ou seja, foi efetuado o *logout* na aplicação e em seguida foram inseridos na barra de endereços do navegador alguns endereços internos, sendo o utilizador direcionado para a página de autenticação da aplicação. Foi ainda testada possibilidade de ter dois separadores no mesmo navegador, com páginas internas da aplicação, e num desses ser efetuado o *logout* da aplicação,

tentando posteriormente continuar a interação no outro separador. Mas tal como no primeiro teste, o utilizador foi direcionado para a página de autenticação.

5.2. TESTE DE *DAEMONS*

O *daemon* `snmpd` é um agente que aguarda pedidos SNMP realizados pela estação de gestão, e que se encontra nos dispositivos a serem monitorizados (GateBoxes). A cada pedido recebido, este *daemon* vai processá-lo e consoante o resultado da análise, irá executar a operação solicitada e enviar uma resposta ao utilizador. O funcionamento do *daemon* `snmpd` foi testado e demonstrado através da execução de um comando SNMP (`snmpget`) na consola do computador que funciona como estação de monitorização com emulação de uma GateBox (máquina virtual), como se pode contactar na Figura 55. Após o pedido obteve-se uma resposta proveniente do agente SNMP (*daemon*) do dispositivo a gerir, onde se podem visualizar três campos: o OID, o tipo de informação recebida e o valor dessa informação.

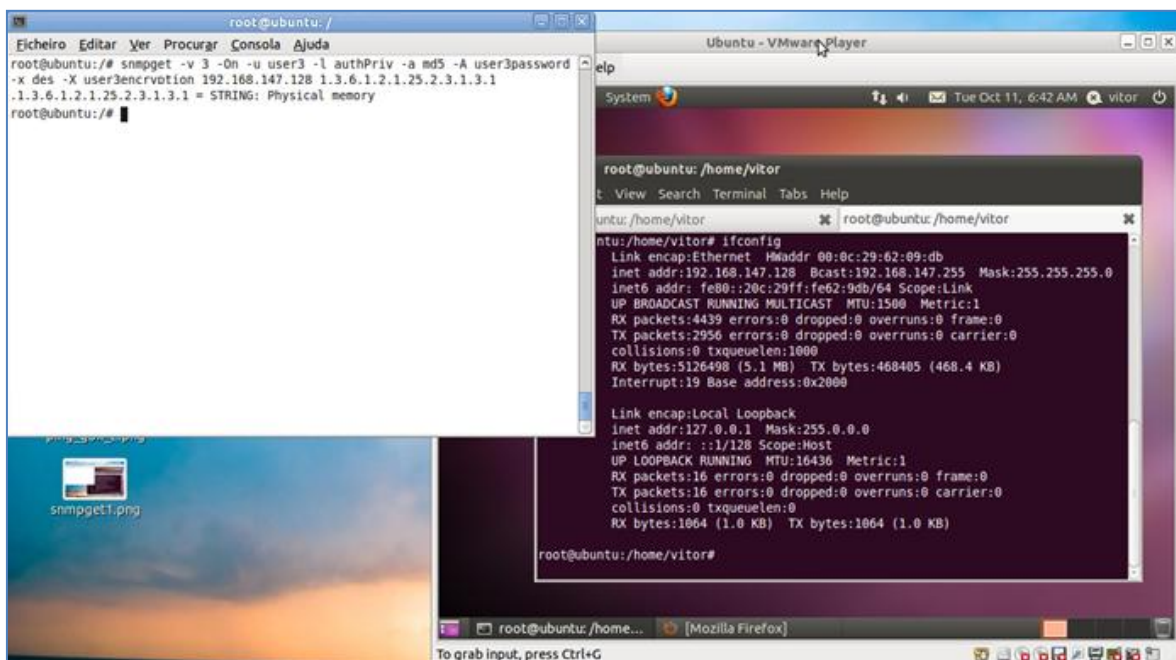



Figura 55 – Demonstração do *daemon* `snmpd`.

Todos os pedidos SNMP recebidos pelo *daemon* `snmpd` são registados num ficheiro de registos (`daemon.log`) da máquina onde está a ser executado. Na Figura 56 pode visualizar-se pedidos efetuados pela estação de gestão (com o IP 192.168.147.1) ao agente do dispositivo monitorizado (com o IP 192.168.147.128). Além disso, pode-se verificar que o pedido efetuado utilizou o protocolo de comunicação UDP.

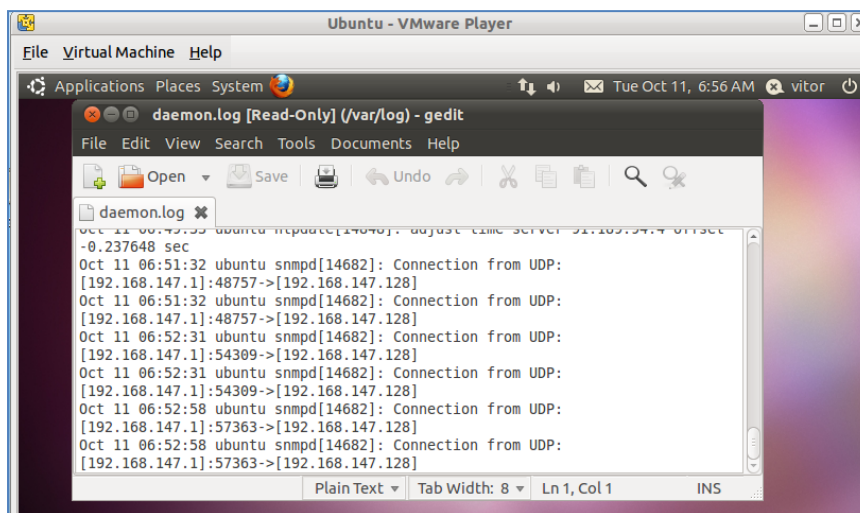



Figura 56 – Ficheiro `daemon.log`.

O *daemon* `snmptrapd` é o responsável pela receção e registo de notificações provenientes das GateBoxes. Para testar e demonstrar o seu funcionamento foi provocada uma notificação, bastando para isso desativar uma interface (vmnet1) da GateBox com o ID igual a 1. Na Figura 57 pode-se visualizar uma sequência de imagens que demonstram a receção e registo de uma notificação no ficheiro (`trap.log`). Inicialmente o *daemon* está ativo e à escuta de alguma notificação que lhe seja enviada. Quando é desativada uma interface de rede no dispositivo monitorizado, este envia um alerta à estação de monitorização. Essa notificação será recebida pelo `snmptrapd` que efetuará o registo da notificação no ficheiro.

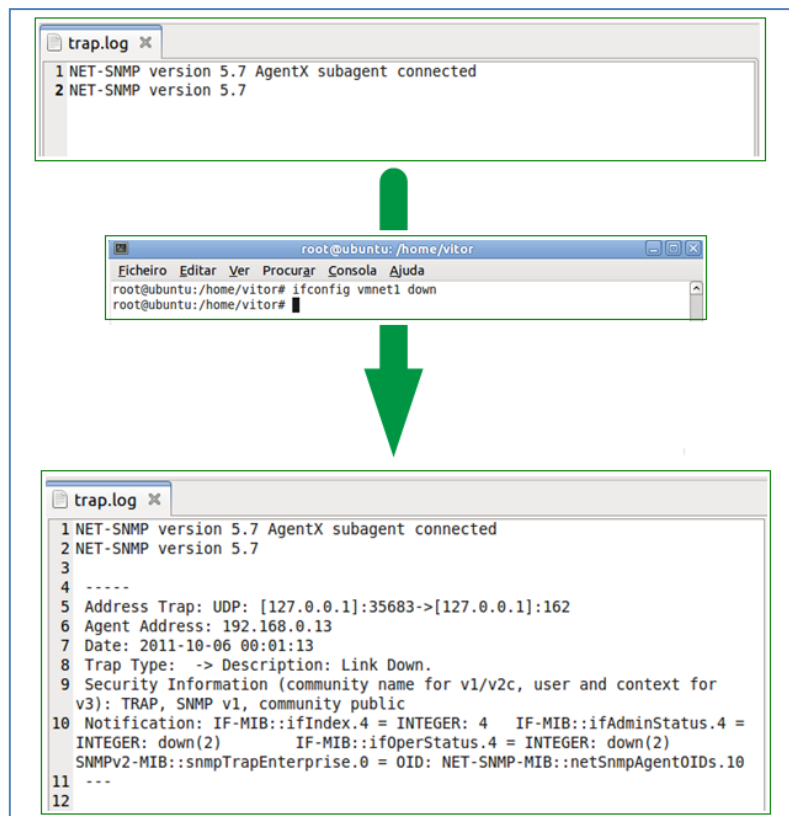



Figura 57 - Demonstração de um registo de uma notificação.

Para demonstração e teste do funcionamento da periodicidade da ativação dos *scripts* desenvolvidos na aplicação, foi configurado o ficheiro do `crontab` para que este execute os *scripts* com um período de um, cinco e dez minutos (Figura 58).

Figura 58 – Configuração do `crontab`.

Após essa configuração realizou-se a análise do ficheiro de registo (`gatekeeper.log`). Dessa forma, constata-se que os *scripts* foram executados periodicamente nos intervalos pretendidos, como se demonstra na Figura 59.

```
2011-10-11 12:34:01.6020 INFO (6): ========== New Request : /GateKeeper/webapp/cron_trap.php ==========
(...)
2011-10-11 12:35:01.7109 INFO (6): ========== New Request : /GateKeeper/webapp/cron_gbx_status.php ==========
(...)
2011-10-11 12:35:01.7115 INFO (6): ========== New Request : /GateKeeper/webapp/cron_trap.php ==========
(...)
2011-10-11 12:36:01.8558 INFO (6): ========== New Request : /GateKeeper/webapp/cron_trap.php ==========
(...)
2011-10-11 12:37:01.9337 INFO (6): ========== New Request : /GateKeeper/webapp/cron_trap.php ==========
(...)
2011-10-11 12:38:02.0222 INFO (6): ========== New Request : /GateKeeper/webapp/cron_trap.php ==========
(...)
2011-10-11 12:39:01.0965 INFO (6): ========== New Request : /GateKeeper/webapp/cron_trap.php ==========
(...)
2011-10-11 12:40:01.2321 INFO (6): ========== New Request : /GateKeeper/webapp/cron_trap.php ==========
(...)
2011-10-11 12:40:01.3014 INFO (6): ========== New Request : /GateKeeper/webapp/cron_gbx_status.php ==========
(...)
2011-10-11 12:40:01.3044 INFO (6): ========== New Request : /GateKeeper/webapp/cron_update.php ==========
(...)
2011-10-11 12:41:01.4579 INFO (6): ========== New Request : /GateKeeper/webapp/cron_trap.php ==========
(...)
2011-10-11 12:43:01.6242 INFO (6): ========== New Request : /GateKeeper/webapp/cron_trap.php ==========
(...)
2011-10-11 12:44:01.7043 INFO (6): ========== New Request : /GateKeeper/webapp/cron_trap.php ==========
(...)
2011-10-11 12:45:01.7866 INFO (6): ========== New Request : /GateKeeper/webapp/cron_trap.php ==========
(...)
2011-10-11 12:45:01.7953 INFO (6): ========== New Request : /GateKeeper/webapp/cron_gbx_status.php ==========
(...)
2011-10-11 12:46:01.9383 INFO (6): ========== New Request : /GateKeeper/webapp/cron_trap.php ==========
(...)
2011-10-11 12:47:02.0226 INFO (6): ========== New Request : /GateKeeper/webapp/cron_trap.php ==========
(...)
2011-10-11 12:48:01.1051 INFO (6): ========== New Request : /GateKeeper/webapp/cron_trap.php ==========
(...)
2011-10-11 12:50:01.2610 INFO (6): ========== New Request : /GateKeeper/webapp/cron_update.php ==========
(...)
2011-10-11 12:50:01.2726 INFO (6): ========== New Request : /GateKeeper/webapp/cron_gbx_status.php ==========
(...)
2011-10-11 12:50:01.2853 INFO (6): ========== New Request : /GateKeeper/webapp/cron_trap.php ==========
(...)
2011-10-11 12:51:01.4841 INFO (6): ========== New Request : /GateKeeper/webapp/cron_trap.php ==========
(...)
2011-10-11 12:56:01.9714 INFO (6): ========== New Request : /GateKeeper/webapp/cron_trap.php ==========
```

De 5 em 5 minutos.

De 10 em 10 minutos.

De 1 em 1 minuto.

Figura 59 – Demonstração da periodicidade do `crontab`.

5.3. TESTE DE *SCRIPTS*

O *script* `cron_update.php` realiza a recolha de informações sobre as GateBoxes, através de pedidos SNMP, e armazena as informações na base de dados. Para testar e demonstrar a execução deste *script*, foi necessário imprimir todos os pedidos SNMP efetuados, assim como as informações a armazenar na base de dados. Na Figura 60 pode-se visualizar alguns pedidos efetuados (`Comand:`). Se o OID correspondente ao pedido efetuado existir na base de dados, o *script* apenas atualiza (`Update ->`) a informação proveniente da resposta. No caso de não existir o *script* cria uma nova linha de informação (`Insert ->`) na tabela para esse OID.

```
Comand: snmpget -v 3 -On -u user3 -l authPriv -a md5 -A user3password -x des -X user3encryption 127.0.0.1 1.3.6.1.2.1.1.1.0
Update -> gbx_id:1 oid_id:1 oid:1.3.6.1.2.1.1.1.0 type: STRING value: Linux ubuntu 2.6.35-28-generic #49-Ubuntu SMP Tue Mar 1 14:39:03 UTC 2011
x86_64

Comand: snmpget -v 3 -On -u user3 -l authPriv -a md5 -A user3password -x des -X user3encryption 127.0.0.1 1.3.6.1.2.1.1.3.0
Update -> gbx_id:1 oid_id:2 oid:1.3.6.1.2.1.1.3.0 type: Timeticks value: (214804) 0:35:48.04

Comand: snmpwalk -v 3 -On -u user3 -l authPriv -a md5 -A user3password -x des -X user3encryption 127.0.0.1 1.3.6.1.2.1.2.2.1.2
Update -> gbx_id:1 oid_id:15 oid:1.3.6.1.2.1.2.2.1.2.1 type: STRING value: lo
Update -> gbx_id:1 oid_id:15 oid:1.3.6.1.2.1.2.2.1.2.2 type: STRING value: eth0
Update -> gbx_id:1 oid_id:15 oid:1.3.6.1.2.1.2.2.1.2.3 type: STRING value: wlan0
Update -> gbx_id:1 oid_id:15 oid:1.3.6.1.2.1.2.2.1.2.4 type: STRING value: vmnet1
Update -> gbx_id:1 oid_id:15 oid:1.3.6.1.2.1.2.2.1.2.5 type: STRING value: vmnet8
```

Figura 60 – Demonstração do *script* `cron_update.php`.

Na aplicação desenvolvida o *script* `cron_status.php` verifica o estado da ligação entre a estação de gestão e o dispositivo a monitorizar. Para demonstrar o funcionamento deste *script* foi efetuado na consola do computador, que desempenha funções de estação de monitorização, um `PING` à GateBox (máquina virtual) (Figura 61). Após ser efetuado o PING é analisada a resposta e armazenado na base de dados o estado de ligação.

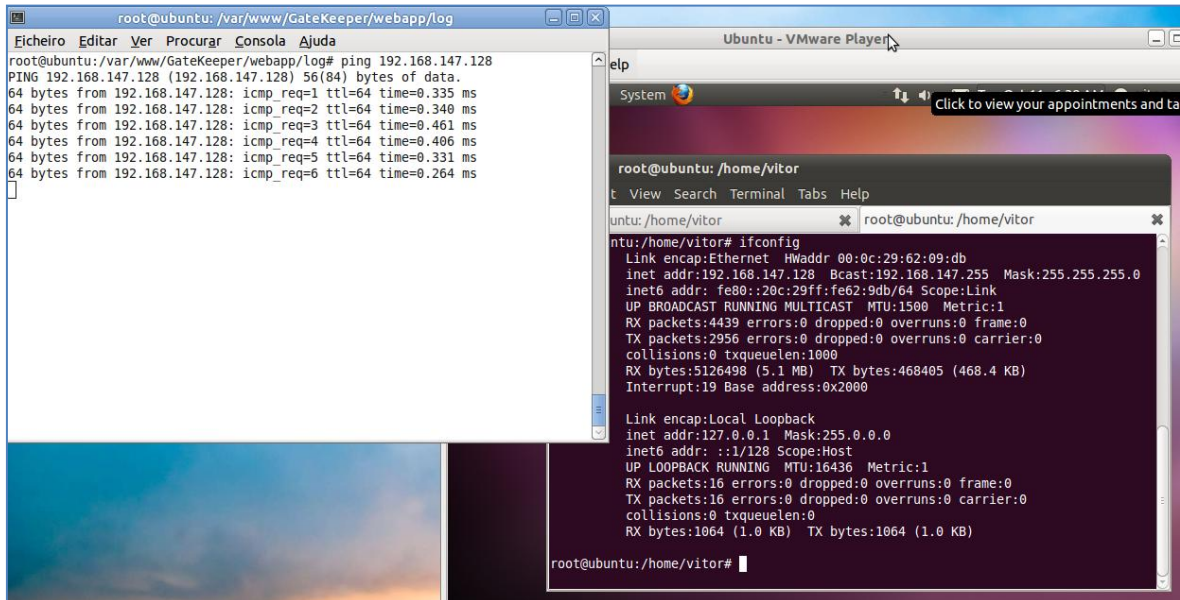



Figura 61 – Demonstração do *script* `cron_status.php`.

O *script* `cron_trap.php` é o responsável pela análise e armazenamento de notificações e ainda pelo envio de *email* para o utilizador. Este é executado periodicamente pelo *crontab*. Na Figura 62 pode-se constatar a interação do *script* com a base de dados, necessária para que este efetue a análise dos dados da notificação registada no ficheiro `trap.log` (1). Após este procedimento são inseridas na base de dados todas as informações relativas a esta notificação (data, tipo de notificação, entre outros) para que possam ser visualizadas na interface Web. Essa inserção é efetuada pela *query* `INSERT INTO` (2).

```
--- Logfile created ---2011-10-05 18:52:07.9993 INFO (6): ========== New Request : /GateKeeper/webapp/cron_trap.php ====
2011-10-05 18:52:07.9995 DEBUG (7): WebCore 0.2.0 started
2011-10-05 18:52:08.0055 DEBUG (7): Template: service running...
2011-10-05 18:52:08.0160 DEBUG (7): DB: created handler 'webcore'
2011-10-05 18:52:08.0167 DEBUG (7): DB: created handler 'gatekeeper'
2011-10-05 18:52:08.0184 DEBUG (7): DB: Query( 1): SELECT DISTINCT * FROM `user` WHERE `user_id`='1'
2011-10-05 18:52:08.0200 DEBUG (7): Icon: service running...
2011-10-05 18:52:08.0209 DEBUG (7): DB: Query( 2): SELECT * FROM `trap` WHERE `select`='Yes'
2011-10-05 18:52:08.0219 DEBUG (7): DB: Query( 3): SELECT * FROM `email` WHERE `select`='Yes'
2011-10-05 18:52:08.0231 DEBUG (7): DB: Query( 4): SELECT * FROM `gbx` WHERE ip_domain='192.168.147.128'
2011-10-05 18:52:08.0241 DEBUG (7): DB: Query( 5): SELECT * FROM `property` WHERE name='NameInterface'
2011-10-05 18:52:08.0252 DEBUG (7): DB: Query( 6): SELECT * FROM `oid` WHERE property_id='15'
2011-10-05 18:52:08.0263 DEBUG (7): DB: Query( 7): SELECT * FROM `data` WHERE gbx_id='2' AND oid='1.3.6.1.2.1.2.2.1.2.2'
2011-10-05 18:52:08.0267 DEBUG (7): DB: Query( 8): SELECT * FROM `trap` WHERE trap_id='1'
2011-10-05 18:52:08.0270 DEBUG (7): DB: Query( 9): SELECT * FROM `trap` WHERE trap_id='2'
2011-10-05 18:52:08.0273 DEBUG (7): DB: Query(10): SELECT * FROM `trap` WHERE trap_id='3'
2011-10-05 18:52:08.0280 DEBUG (7): DB: Query(11): SELECT gbx_id FROM `event` WHERE date =' 2011-08-01 19:32:44
        ' AND gbx_id='2'
2011-10-05 18:52:08.0286 DEBUG (7): DB: Query(12): INSERT INTO `event` (date,gbx_id,type,inform,notification)
        VALUES (' 2011-08-01 19:32:44  \n ','2','Cold Start','Interface  eth0 down','Address Trap: UDP:
        [192.168.147.128]:57662->[192.168.147.1]:162 \n Agent Address: 0.0.0.0 \n Date: 2011-08-01 19:32:44
        \n Trap Type: 0 -> Description: Cold Start. \n Security Information (community name for v1/v2c,
        user and context for v3): TRAP2, SNMP v3, user traptest, context  \n Notification:
        DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (0) 0:00:00.00 SNMPv2-MIB::snmpTrapOID.0 = OID:
        IF-MIB::linkDown     IF-MIB::ifIndex.2 = INTEGER: 3  IF-MIB::ifAdminStatus.2 = INTEGER: down(2) \n ')
2011-10-05 18:52:08.0290 DEBUG (7): DB: Query(13): SELECT * FROM `gbx` WHERE gbx_id='2'
2011-10-05 18:52:08.0293 DEBUG (7): DB: Query(14): SELECT * FROM `email` WHERE `select`='Yes'
2011-10-05 18:52:08.0297 DEBUG (7): DB: Query(15): SELECT * FROM `email` WHERE email_id='1'
```

Figura 62 – *Log* da inserção de uma notificação na base de dados.

As notificações recebidas são encaminhadas ao utilizador através do envio de um *email*. Esse envio é efetuado pelo script `cron_trap.php`. Na Figura 63 visualiza-se a informação enviada no *email*, a informação geral da notificação (após a análise desta), da data da ocorrência, o ID e o nome da GateBox. Além disso, também é enviado a *Trap Notification* do evento.

Figura 63 – *Email* de notificação.

No caso de receção de uma notificação em que se desconheça a sua origem (GateBox) é apenas alertado o utilizador com um *email* (Figura 64), não sendo armazenada na base de dados esta notificação.

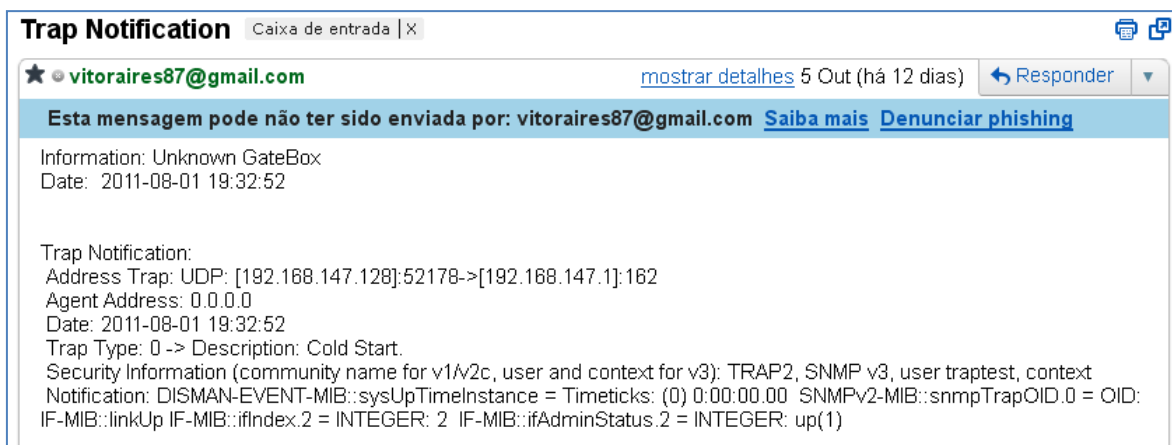



Figura 64 – *Email* de notificação de uma GateBox desconhecida.

Os resultados de todos os testes realizados permitiram concluir que, apesar de se tratar de um protótipo, a aplicação GateKeeper pode desempenhar com sucesso as tarefas para as quais foi desenvolvida.

6. CONCLUSÕES

O trabalho realizado teve como objetivo o estudo e implementação de uma aplicação Web que permitisse uma interação com os sistemas GateBox desenvolvidos pela empresa NextToYou. O desenvolvimento da aplicação teve como requisitos principais a monitorização de informação, assim como a receção e análise de alarmes e avisos gerados pelas GateBox.

O sistema desenvolvido (GateKeeper) apresenta um conjunto de requisitos definidos. Uma dessas funcionalidades é o facto de proporcionar ao utilizador uma forma de interação com a aplicação através de uma interface Web. Através dessa interface é possível a identificação dos sistemas a monitorizar, bem como visualizar a informação com eles relacionados (características, informações obtidas pelo protocolo SNMP e o estado da ligação). A visualização das notificações enviadas pelos dispositivos no caso de ocorrer algum evento é, também, facultada na interface. Além disso, é proporcionada ao utilizador uma área de configurações de parâmetros, necessários na monitorização e notificação, tais como, a definição do perfil SNMP utilizado por cada grupo de GateBox, a configuração dos tipos de notificação e dos *emails* para o envio de alertas ao utilizador. Por fim, uma das características mais relevantes é a possibilidades de definição de OIDs na interface, oferecendo ao utilizador da aplicação uma monitorização mais específica de determinados objetos. Para cada um dos aspetos a interface proporciona diferentes funcionalidades, o que

torna o sistema flexível e de utilização intuitiva, sendo algumas: a consulta, a inserção, a modificação e a eliminação de parâmetros. Outro requisito do sistema Gatekeeper é a independência da aplicação Web ativa para o seu funcionamento. Assim sendo, foram criados e configurados no servidor, *scripts* e *daemons* de modo a garantir a monitorização dos dispositivos. Estes desempenham funcionalidades de recolha e armazenamento de informação das GateBoxes através do protocolo SNMP, assim como de verificação do estado de ligação do dispositivo, e ainda de receção de notificações geradas pelos dispositivos monitorizados, com consequente envio de um alerta para o utilizador. Por fim, o sistema GateKeeper integra uma base de dados, onde a informação definida na interface Web é armazenada, bem como a informação obtida na execução dos *scripts* no servidor.

De forma a validar o funcionamento do sistema Gatekeeper foram realizados alguns testes e demonstrações para cada um dos aspetos principais do sistema: a aplicação Web, a interação com a base de dados, os *scripts* e os *daemons*. Através destes testes e demonstrações, conclui-se que o sistema tem um bom funcionamento e que cumpre todos os requisitos pretendidos. Da análise da aplicação Web pode-se afirmar que proporciona ao utilizador uma interface simples, intuitiva e eficaz. Assim sendo, a interface cumpre todas as normas básicas de criação de uma aplicação Web, tais como, a apresentação do menu principal em todas as páginas, e transmissão da informação de uma forma clara e lógica, sem excesso de conteúdos. Em termos dos testes de funcionalidades, as comunicações com a base de dados foram avaliadas em diferentes operações: consulta, inserção, modificação e eliminação de informação. Cada um destes aspetos foi avaliado com sucesso. No teste de interface foram testados os servidores que servem de suporte à aplicação Web, o servidor Web e o servidor de base de dados. Para isso foi analisado o comportamento destes servidores em caso de falha na conexão. Tal como era espectável a aplicação Web não é necessariamente executada e, para cada caso, o utilizador é alertado com uma mensagem. A compatibilidade da aplicação Web é um aspeto importante para o utilizador visto poder afetar o bom funcionamento da aplicação. Assim, foi necessário realizar testes a todas as funcionalidades da interface da aplicação em diferentes *browsers*, não sendo detetada nenhuma anomalia em termos funcionais. Em termos de segurança, a verificou-se que qualquer interação só pode ser efetuada após a autenticação, sendo criada uma sessão de utilizador para que este interaja nas diferentes páginas da interface. Por último, o desempenho da aplicação Web foi avaliado num ambiente emulado constituído por um computador local, uma máquina virtual (VMware) e um computador externo. Deste modo,

foram obtidos tempos de execução da funcionalidade do menu `Update` da aplicação Web que realiza a recolha e armazenamento de informação de GateBoxes através do protocolo SNMP. Essa execução é igual à realizada pelo *script* `cron_update.php`. Após a análise dos tempos para uma GateBox (computador externo), pode constatar-se que um servidor com as características do utilizado no teste não seria o mais eficaz para a execução da aplicação, pois os tempos de resposta obtidos podem tornar-se demasiados elevados para um número significativo de GateBoxes. Contudo, ao executar a aplicação desenvolvida num servidor dedicado, em relação ao utilizado, os tempos serão reduzidos e assim melhorada a *performance* da aplicação. Os testes e demonstrações realizados para os *scripts* e *daemons* criados e configurados para o sistema evidenciaram um bom funcionamento.

Em termos funcionais, a aplicação desenvolvida corresponde aos objetivos estabelecidos, tornando-se assim numa mais-valia para a empresa. No entanto, podem ser feitos alguns melhoramentos tais como, em termos de visualização gráfica através da utilização por exemplo de CSS que vai tornar a aplicação mais apelativa e intuitiva. Outro melhoramento seria a adição de um suporte para gráficos em tempo real onde o utilizador pudesse visualizar por exemplo, o desempenho da CPU de um dos dispositivos. Além destes, na consulta de GateBoxes a introdução de um mecanismo de pesquisa, permitiria ao utilizador filtrar a sua busca pelo nome, morada, categoria ou localidade. Por fim, outro melhoramento seria que o sistema possibilitasse a alteração remota dos valores dos OIDs, nos dispositivos a monitorizar, através do protocolo SNMP.

Referências Documentais

[1] STALLINGS, William — Redes e Sistemas de Comunicação de dados, 5ª edição, Editora Campus. 2005. ISBN: 85-352-1731-2

[2] ROCHA, Ivandro José de Freitas e DOURADA, Marcelo Oliveira Serra — (Trabalho de conclusão de curso) Gerência de redes de computadores utilizado o Zabbix: um estudo em caso. Universidade Católica de Goiás, Brasil, Junho, 2008. http://aldeia3.computacao.net/greenstone/collect/trabalho/index/assoc/HASHd0ca.dir/doc.pdf

[3] SILVEIRA, Rafael van de Sandre — (Trabalho de conclusão de curso) Processo de planeamento para elaboração de política de gerenciamento de rede para micro e pequenas empresas. Universidade do Vale de Itajaí, São José, Dezembro de 2004, Brasil. http://siaibib01.univali.br/pdf/Rafael%20Silveira.pdf

[4] NextToYou, http://www.nexttoyou.pt/

[5] ALVES, Rogério Furlanetti — (Monografia de Curso de Especialização) Gerência de rede utilizando software livre, Universidade Estadual londrina, Paraná, 2007, Brasil. http://www2.dc.uel.br/nourau/document/?view=551

[6] PINHEIRO, José Maurício dos Santos— (Artigo) Gerenciamento de Redes de Computadores , 3 de Julho de 2006. http://www.projetoderedes.com.br/artigos/artigo_gerenciamento_de_redes_de_computadores.php

[7] PINHEIRO, José Maurício dos Santos — Gerenciamento de Redes de Computadores versão 2.0, Agosto 2002. http://www.allnetcom.com.br/upload/GerenciamentodeRedes.pdf

[8] Cron Jobs, http://en.wikipedia.org/wiki/Cron

[9] FRAGA, Sandro David Ribeiro — (Dissertação) Monitorização de Processos Multimédia, Faculdade de Engenharia da Universidade do Porto, Porto. 2008. http://www.dart-europe.eu/full.php?id=281289

[10] MAURO Douglas and SCHMIDT Kevin — Essential SNMP, 2nd Edition, O'Reilly, September 2005. ISBN: 0-596-00840-6

[11] STALLINGS, William — Network Security Essentials: Applications and standards, 2nd Edition, Prentice Hall. ISBN 0-13-035128-8.

[12] MIB Information, http://penta.ufrgs.br/gr952/trab1/2mibII.html

[13] ALVARENGA, Igor Drummond e RAMOS, Bruno Lange — Simple Network Management Protocol (SNMP), Universidade Federal do Rio de Janeiro, Brasil. http://www.gta.ufrj.br/grad/11_1/snmp/

[14] MILLER, Mark A. — Managing Internetworks with SNMP, Second Edition, M&T Books. ISBN: 1558515615

[15] DELFINO, Gardel Moreira — SNMP- Simple Network Management Protocol, Universidade Federal do Rio de Janeiro, Brasil. http://www.gta.ufrj.br/grad/98_2/gardel/snmpv3.html

[16] ANDRADE, Hetty Alves — (Monografia de Pós-Graduação) Nagios como solução de monitorizamento de rede. Lavas, 2006. http://www.ginux.ufla.br/files/mono-HettyAndrade.pdf

[17] Nagios, http://www.nagios.org/

[18] Nagios Documentation, http://nagios.sourceforge.net/docs/nagios-3.pdf

[19] Nagios Images, http://www.smartmon.com.au/images/docs/Nagios/

[20] BONOMO, Esley — (Monografia de Pós Graduação) Gerenciamento e monitorização de redes de computadores utilizando-se Zabbix, Universidade Federal de Lavras, Brasil, 2006. http://www.ginux.ufla.br/files/mono-EsleyBonomo.pdf

[21] Cacti, http://www.cacti.net/index.php

[22] KUNDU, Dinangkur and LAVLU, S. M. Ibrahim — Cacti 0.8 Network Monitoring. Packt Publishing Ltd, ISBN: 978-1-847195-96-8.

[23] Cacti Documentation, http://pt.scribd.com/doc/7234195/Cacti

[24] Ubuntu, http://pt.wikipedia.org/wiki/Ubuntu

[25] Ubuntu Documentation, http://sites.google.com/site/sistemaoperativognulinuxubuntu/ubunto/caracteristicas

[26] Zabbix, http://www.zabbix.com/

[27] Zabbix Imagem, http://upload.wikimedia.org/wikipedia/commons/9/99/Zabbix.png

[28] Zabbix Imagem, http://dbzer0.com/wp-content/uploads/2008/02/zabbix_routers_map_smaller.png

[29] PEAR, http://pear.php.net/manual/

[30] Smarty Documentation, http://www.smarty.net/docs/en/

[31] Synfony, http://www.symfony.com/

[32] Zend, http://framework.zend.com/manual/en/

[33] Daemon Definition, http://www.linfo.org/daemon.html

[34] PALNI, Gowri Shankar — (Article) Summary of web application testing methodologies and tools. WebSphere Software Group, IBM, 29 Mar 2011. http://www.ibm.com/developerworks/web/library/wa-webapptesting/index.html?ca=drs-

[35] KOTA, Kristen — (Article) Testing Your Web Application: A Quick 10-Step Guide. 2005. http://www.adminitrack.com/articles/TestingYourWebApps.aspx

[36] Web Testing: Complete guide on testing Web applications, http://www.softwaretestinghelp.com/web-application-testing/

[37] Google Maps API, http://code.google.com/intl/pt-PT/apis/maps/index.html

[38] Google Maps API Examples, http://code.google.com/intl/pt-PT/apis/maps/documentation/javascript/v2/examples/index.html

[39] Web Testing:Complete Guide To Test Your Web Applications, http://www.guru99.com/web-application-testing.html

[40] Monitorização de redes, http://file.tecnolowellington.webnode.com.br

[41] LEHMANN, Erny Otto — Introdução ao CMIP, Universidade Federal do Rio de Janeiro, Brasil. http://www.gta.ufrj.br/grad/cmip.html#cmip

[42] The Internet Engineering Task Force (IETF), http://www.ietf.org/

[43] Remote Operation Service Element (ROSE), http://www.teleco.com.br/tutoriais/tutorialosi/pagina_6.asp

[44] Associotion Control Service Element (ACSE), http://www.coursework.biz/Essays/University/Computer_Science/295/

[45] SERRÃO, Carlos e MARQUÊS, Joaquim — Programação com PHP 5.3, FCA – Editora de Informática. Outubro de 2009. ISBN: 978-972-722-341-1

[46] ANACOM, www.anacom.pt/

[47] CONTESSA, Diego Fraga e POLINA Everton Rafael — (Artigo) Gerenciamento de equipamentos usando o protocolo SNMP, Departamento de pesquisa e desenvolvimento – CP Electrônica S.A. Porto Alegre, Brasil.

[48] MAIA, João Prado, HAYDER, Hasin and GHEORGHE, Lucian — Smarty: PHP Template programming and Applications. Packt Publishing. 2006. ISBN: 1-904811-40-X

[49] Smarty Information, http://bloghoctap.com/web-design/smarty-mot-cong-cu-thiet-yeu-cho-nguoi-lam-web.html

[50] Dependency Injection Container , http://fabien.potencier.org/article/11/what-is-dependency-injection

[51] Model-View-Controller, http://msdn.microsoft.com/en-us/library/ff649643.aspx